FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Exploiting new types of structure for fixed-parameter tractability

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Mgr. Eduard Eiben
Registration Number 1428756

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dr. Stefan Szeider
Second advisor: Prof. Dr. Georg Gottlob

The dissertation has been reviewed by:

Saket Saurabh	Henning Fernau

Vienna, 9th April, 2018

Eduard Eiben

Erklärung zur Verfassung der Arbeit

Mgr. Eduard Eiben

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, 9. April 2018

Eduard Eiben

Acknowledgements

I am very grateful for the opportunity to study at TU Wien under the supervision of Stefan Szeider. Stefan was a great mentor who showed me how to navigate through the academia and also gave me a lot of freedom to pursue the research topics that interest me. I am also very thankful to Robert Ganian, who was my closest collaborator and an unofficial advisor, and to Sebastian Ordyniak who recommended me to apply for this position and whom I enjoyed working with a lot. Next, I thank all my co-authors for introducing me to many new interesting questions and for all the hours spend staring at the whiteboard together.

I greatly appreciate also the possibility to be a part of DK LogiCS. The program provided me with a lot of support and many new friends. Furthermore, I thank all my friends and colleagues from the university for all the coffee breaks, lunches, and all the diversions from the work that kept me sane. I thank all my closer friends and the people from Ultimate Frisbee community in Vienna for all the distractions outside the university and for making me miss my PhD life in Vienna already.

Last but not least, I am very grateful to my parents for all the support and encouragement to follow my passion throughout all my life.

Abstract

The design of efficient algorithms for various problems is a fundamental part of computer science. One significant obstacle in this area is the fact that many interesting problems from areas such as Algorithmic Graph Theory, Artificial Intelligence, or Optimization are known to be NP-hard on general instances. However, in the majority of applications the instances of interest are the result or output of some other processes, and as such inherently contain some form of structure. This structure can often be exploited to efficiently find exact solutions for many NP-hard problems.

To capture the structure of the input instance, we use the framework of parameterized complexity. In parameterized algorithms, the running time is analyzed in finer detail than in classical complexity theory: instead of expressing the running time as a function of the input size only, dependence on a parameter k is taken into account. We use the parameter k to describe how "structured" the input instance is. Algorithms with running time $f(k)n^c$, where n is the input size and c is a constant independent of both n and k, are called *fixed-parameter algorithms* or *FPT algorithms*. Typically the goal in parameterized algorithms is to design FPT algorithms while minimizing both the $f(k)$ factor and the constant c in the bound on the running time.

We study a range of different problems under the paradigm of parameterized complexity. More precisely, our research focuses on two approaches to analyze the structure of instances and their combination. The first approach focuses on the notion of *decomposition*. The idea is to decompose the given input into simpler parts and use this decomposition to solve the problem more efficiently. The second approach is based on the established notion of a *modulator* to a tractable class of instances, which applies to problem instances that may be placed in a tractable class by a small number of local changes.

We design several algorithms that exploit some of the already established parameters. However, our main focus is on designing and exploiting new kinds of structure. We highlight here one of the results in particular. We develop a family of parameters that *combine* the two approaches outlined above to capture forms of structure not accessible to a single approach, and we show that these parameters give rise to efficient algorithms for a plethora of NP-hard problems.

Kurzfassung

Eine fundamentaler Aufgabe der Informatik ist die Entwicklung effizienter Algorithmen für Probleme, die in verschiedenen Bereichen auftreten. Eine wesentliche Herausforderung hierfür stellt die Tatsache dar, dass viele wichtige Probleme aus Bereichen wie algorithmischer Graphentheorie, künstlicher Intelligenz, und Optimierung für allgemeine Problemeingaben NP-schwer sind. Jedoch sind in vielen Anwendungen die relevanten Problemeingaben das Resultat von Prozessen und weisen daher eine gewisse Form von Struktur auf. Diese Struktur kann oft ausgenutzt werden um NP-schwere Problem effizient zu lösen.

Um die Struktur in Problemeingaben zu erfassen, verwenden wir den theoretischen Rahmen der parametrisierten Komplexitätstheorie. Die Laufzeit parametrisierter Algorithmen wird in feinerem Detail analysiert als es die klassischen Komplexitätstheorie gestattet: anstatt die Laufzeit nur als Funktion der Eingabegrösse zu bestimmen, wird hier auch die Abhängigkeit von einem Parameter k berücksichtigt. Der Parameter dient dazu, den Grad anzugeben, wie sehr die Eingabe eine Struktur aufweist. Algorithmen mit einer Laufzeit von $f(k)n^c$, wobei n für die Eingabegrösse und c für eine Konstante stehen, werden als *fest-Parameter-handhabbar (fixed-parameter tractable, FPT)* bezeichnet. Das Ziel in der parameterisierten Komplexitätstehorie ist es, FPT-Algorithmen zu entwickeln, und dabei sowohl den Faktor $f(k)$ als auch die Konstante c zu minimieren, um eine möglichst geringe Laufzeit zu erhalten.

Wir untersuchen eine breite Palette von unterschiedlichen Problemen unter dem Paradigma der parameterisierten Komplexitätstheorie. Hierbei legen wir besonderes Augenmerk auf zwei fundamentale Ansätze, um die Struktur in Probleminstanzen zu erfassen und algorithmischen auszunutzen: Der erste Ansatz zielt auf die *Zerlegbarkeit* von Problemeingaben ab. Die Grundidee ist es, die Problemeingabe in einfache Teile zu zerlegen, aus deren Lösungen eine globalen Lösung erstellt werden kann. Der zweite Ansatz basiert auf dem fundamentalen Begriff des *Modulators* in eine handhabbare Klasse. Hier ist die Grundidee, dass die Problemeingabe nach bestimmten lokalen Modifikationen in eine als handhabbar bekannte Klasse von Problemeingaben fällt.

Wir entwickeln verschiedene Algorithmen für bereits bekannte Parameter. Unser Hauptaugenmerk liegt aber in der Entwicklung von neuen Parametern, welche das Ausnützen von neunen Formen von Struktur ermöglichen. Zum Beispiel entwickeln wir eine Reihe von Parametern welche die *Kombination* der beiden beschriebenen Ansätze darstellen. Hierbei können strukturelle Eigenschaften in Problemeingaben ausgenützt werden, die nicht durch einen der beiden Ansätze alleine erfassbar sind. Wir zeigen, dass diese Parameter zu FPT-Algorithmen für eine Fülle von NP-schweren Problemen führen.

Contents

CHAPTER 1

Introduction

The notion of *computation* has existed in some form for thousands of years. In its everyday meaning, this term refers to the process of producing an output from a set of inputs in a finite number of steps and some kind of computational processes arise in many different disciplines and also in nature. We focus on issues of *computational efficiency*. Computational complexity theory asks the following simple question: how much computational resources are required to solve a given computational task? A very important limitation to solving the task is the time needed to obtain the answer. We would like to know how long our computation takes before we assign resources towards computing a task that might not terminate in our lifetime. This leads us to one of fundamental parts of computer science that is the design of efficient algorithms. To formalize the notion of efficient algorithm, Cobham and Edmonds in 1965 identified in two separate papers the tractable problems to be exactly the problems in the complexity class P, that is the ones solvable in polynomial time. One significant obstacle here is the fact that many interesting problems from areas such as Artificial Intelligence, Algorithmic Graph Theory, Optimization, Computational Logic are known to be NP-*hard* on general instances and therefore believed to not allow an algorithm running in polynomial time. However, in the majority of applications the instances of interest are the result or output of some other processes, and as such inherently contain some form of structure. This structure can often be exploited to efficiently find exact solutions for many NP-hard problems.

We use the well-established framework of parameterized complexity, introduced by Downey and Fellows [63], to capture the structure of instances. In this framework, we add to each input a *parameter* k, which is simply a relevant secondary measurement that encapsulates some aspect of the input instance, be it the size of the solution sought for or a number describing how “structured” the input instance is. The smaller the parameter k is, the more structured is the input instance. Therefore, the goal in *parameterized algorithms* is to design an algorithm which performs well if the parameter k is small. It turns out that a good definition for a tractable algorithm in the parameterized complexity framework are the algorithms with running time $f(k)n^c$, where n is the input size and c is a constant independent of both n and k. Such algorithms are called *fixed-parameter algorithms*, or shortly *FPT algorithms*.

However, sometimes it is observed with dismay that every proposed algorithm to solve a given problem is not fixed-parameter tractable. In classical complexity, the theory of NP-completeness gives us one common obstacle to tractability, and hence offers a conditional lower bound on the running time for any NP-hard problem. The parameterized complexity approach adapts this pragmatic viewpoint to algorithm design and develops a similar lower bound theory. In many cases, it is very helpful to look at a problem from both the algorithmic and the complexity viewpoint at the same time. A failed attempt at finding an algorithm can highlight certain difficult situations, giving insight into the structure of hard instances, which can be the starting point of a hardness proof.

1.1 General Goals

The general aim of this thesis is to study and exploit the structure of the input to solve problems that are NP-hard in the general case. As our main tool we use to achieve this is the parameterized complexity framework. We focus our research on two prominent approaches to analyze the structure of instances and on possibilities of their combination. The first approach focuses on the notion of *decomposition*. The idea is to decompose the given input into simpler parts and use this decomposition to solve the problem more efficiently. The second approach is based on the established notion of a *modulator* to a tractable class of instances, which applies to problem instances that may be placed in a tractable class by a small number of local changes. In addition, we study the possibilities to *combine* these two approaches to capture the forms of structure not accessible to a single approach.

We target various problems that naturally appear in the areas of Graph Theory, Artificial Intelligence, and related fields and we highlight here three general goals we consider when targeting a specific problem.

1. *Identify useful problem-specific structure*—this structure should not only be helpful to design more efficient algorithms, but also efficiently recognizable;

2. *Design FPT algorithms*—once we identify structure that might be useful, we want to exploit it to obtain efficient algorithms with good worst-case running time guaranties; and

3. *Develop matching lower bounds*—either showing that an FPT algorithm does not exist, or establishing that the obtained running times cannot be improved (under established complexity assumptions).

1.2 Problems in Consideration

In this thesis, we investigate how the structure of the given instance can help to solve a variety of NP-hard problems more efficiently. This section serves to briefly introduce the problems we consider in our thesis.

1.2.1 Counting problems

Counting the number of linear extensions of a partially ordered set (poset), or shortly #LinExt, is a fundamental problem of order theory that has applications in a variety of distinct areas such as sorting [171], sequence analysis [154], convex rank tests [157], sampling schemes of Bayesian networks [163], and preference reasoning [153]. Determining the exact number of linear extensions of a given poset is known to be #P-complete [34] already for posets of height at least 3. Informally, #P-complete problems are as hard as counting the number of accepting paths of any nondeterministic Turing machine, implying that such problems are not tractable unless P = NP.

1.2.2 Vertex deletion problems

Vertex deletion problems ask whether it is possible to delete at most k vertices from a graph so that the resulting graph belongs to a specified graph class. Over the past years, the parameterized complexity of vertex deletion to a plethora of graph classes has been systematically researched. We study two different vertex deletion problems.

Directed Feedback Vertex Set (DFVS) is the problem of finding a set of k vertices that intersects all directed cycles in a given digraph. For over a decade resolving the fixed-parameter tractability of DFVS was considered the most important open problem in parameterized complexity. In fact, this problem was posed as an open problem in the first few papers on fixed-parameter tractability [60, 62]. DFVS was shown to be fixed-parameter tractable in a breakthrough paper by Chen, Liu, Lu, O'Sullivan and Razgon [42] in 2008.

Distance-Hereditary Vertex Deletion is the vertex deletion problem where the target class is the class of distance-hereditary graphs. A graph is distance-hereditary if the distances between vertices are preserved in all connected induced subgraphs of the original graph. Since its introduction by Howorka [119] in 1977, this graph class has been extensively studied and several otherwise NP-hard problems has been shown to be polynomial time solvable on the distance-hereditary graphs [44, 56, 123, 121, 124, 192, 158, 101]. Additionally, distance-hereditary graphs are precisely all graphs with *rank-width* at most 1 [167] and as a consequence, by Courcelle's theorem, efficient dynamic programming algorithms exist for many problems on these graphs [50].

1.2.3 Meta-problems

To be able to cope with many problems at once we also target several problems in computer science that, by their nature, make it easier to encode other problems. An efficient algorithm for such *"meta-problem"* gives us a good framework for solving many problems. Namely, once we have a efficient algorithm for such a problem, we only need to find an encoding of our problem and run the already implemented efficient algorithm on the encoding. In the following we introduce three such problems that are often used in practice to design algorithm for NP-hard problems.

Integer Linear Programming (ILP) is an archetypical representative of NP-complete optimization problems and has a broad range of applications in various areas of artificial intelligence. In particular, a wide variety of problems in Optimization and Artificial Intelligence are efficiently solved in practice via a translation into an ILP, including problems from areas such as planning [196, 197], process scheduling [84], packing [149], vehicle routing [194], and network hub location [10]. For example, in Figure 1.1 we give an encoding for the well-know NP-hard problem Vertex Cover, which is the vertex deletion problem to the class of edgeless graphs.

$$\begin{array}{ll}
\mathtt{minimize} \sum_{\mathtt{v}\in\mathtt{V}} \cdot \mathtt{x}_{\mathtt{v}}, & \text{where} \\
\mathtt{x}_{\mathtt{u}} + \mathtt{x}_{\mathtt{v}} \geq \mathtt{1} & \forall uv \in E, \\
\mathtt{x}_{\mathtt{v}} \geq \mathtt{0};\ \mathtt{x}_{\mathtt{v}} \in \mathbb{Z} & \forall v \in V.
\end{array}$$

Figure 1.1: ILP encoding the Vertex Cover problem for a graph $G = (V, E)$.

The problem of *evaluating quantified Boolean formulas*, called also shortly *QBF*, is the archetypical PSpace-complete problem and is therefore believed to be computationally harder than the NP-complete problems such as ILP or SAT, which is the problem of determining the satisfiability of a propositional formula and is closely related to QBF [140, 169, 189]. Moreover, many important computational tasks such as verification, planning, and several questions in knowledge representation and automated reasoning can be naturally encoded as a QBF instance [69, 166, 175, 182]. In recent years quantified Boolean formulas have become a very active research area.

$$\forall a \exists b \exists c \forall d (a \vee \neg b \vee d) \wedge (\neg c \vee \neg a \vee d) \wedge (\neg b \vee c \vee \neg d)$$

Figure 1.2: An example of a QBF formula.

The *model checking problem*, i.e., the problem to decide whether a given logical sentence is true in a given structure, is a fundamental computational problem which appears in a variety of areas in computer science, see for example [93], [26], or [97]. This problem in its full generality is computationally intractable. However, if we fix a logical sentence in first order logic, the model checking problem becomes polynomially tractable. Hence, to express NP-hard problems we need more powerful logics. A prominent logic often used in this setting is Monadic Second Order (MSO) logic [50] along with its extensions such as Counting MSO and Optimization MSO. These provide a powerful tool capable of expressing many NP-hard problems with constant size formulas. For example, Figure 1.3 gives a constant length formula expressing that the input graph is 3-colorable.

$$\exists X \exists Y \exists Z \{\forall x \in V(x \in X \vee x \in Y \vee x \in Z) \wedge \forall x, y \in V(edge(x,y) \rightarrow \\ [\neg(x \in X \wedge y \in X) \wedge \neg(x \in Y \wedge y \in Y) \wedge \neg(x \in Z \wedge y \in Z)])\}$$

Figure 1.3: MSO formula encoding the 3-Coloring problem for a graph $G = (V, E)$.

1.3 Background

In order to properly explain the techniques and results in this thesis, we first give a very brief and somewhat informal introduction to the most important notions in parameterized complexity.

1.3.1 Parameterized Complexity

Many useful problems in computer science are known to be NP-hard, for instance Vertex Cover (given a graph G and an integer k, does the graph contain a vertex cover of k vertices?) [98]. Thus, by standard complexity theory we expect that any exact solution algorithm will run in exponential time.

The problem can be solved in worst-case time $O(n^k)$, simply by enumerating all k-element subsets of the input graph. On classes of inputs where k is much smaller than n, a solution algorithm with runtime of, e.g., $2^k \cdot n^2$ would vastly outperform the runtime of the previous algorithm. For example, solving an instance of size $n = 1000$ with $k = 10$ may still be considered feasible with an algorithm requiring $2^k \cdot n^2$ operations, but an n^k algorithm would in this case require more operations than there are stars in the observable universe.

This observation motivated the introduction of *parameterized problems*. A parameterized problem is a computational problem whose instances have two parts, an *input*, say x (of size n), and a *parameter*, say k. A parameterized problem which can be solved in time $f(k) \cdot n^d$, where d is a constant independent of k, is called *fixed-parameter tractable* (FPT in short). Similarly, an algorithm running in time $f(k) \cdot n^d$ is called an *FPT algorithm*. The class FPT is then the class containing all problems that admit an FPT algorithm. However, just like we cannot hope to obtain polynomial-time algorithms for all problems in the classical complexity paradigm, in the parameterized paradigm we will not always be able to obtain an FPT algorithm. In these cases obtaining an algorithm which runs in time n^k is still more desirable than an exponential algorithm running in time 2^n. Therefore, especially when we can show that an FPT algorithm is unlikely, we will try to obtain at least so-called XP algorithms, that is, an algorithm running in time $n^{f(k)}$. The class XP then contains all problems that have such XP algorithm.

A great benefit of the parameterized notion of tractability is that a single problem can have a variety of parameterizations, and the choice of the parameterization affects its parameterized complexity. The example above uses the solution size as the parameter, and so an FPT algorithm for, e.g., Vertex Cover will remain efficient on all graphs as long as we only search for small vertex covers. However, we may also use the parameter to capture the structure contained in our input graphs; these so-called *structural parameters* have become a focal point of research in parameterized complexity. Structural parameters such as *treewidth* [177] and *rank-width* [118] allow the efficient solution of a wide range of problems regardless of their solution size (see for example [46, 93]); instead, the running time depends on how "well-structured" the instances are.

1.3.2 Kernelization

Kernelization is another algorithmic technique that has become the subject of a very active field in parameterized complexity, see, e.g., the references in [77, 110, 180]. Kernelization can be considered as a *preprocessing with performance guarantee* that reduces an instance of a parameterized problem in polynomial time to a decision-equivalent instance, the *kernel*, whose size is bounded by a function of the parameter alone. Once a kernel is obtained, the time required to solve the original instance is bounded by a function of the parameter and therefore independent of the input size. Consequently one aims at kernels that are as small as possible.

Every fixed-parameter tractable problem admits a kernel, but the size of the kernel can have an exponential or even non-elementary dependence on the parameter [63, 86]. Thus research on kernelization is typically concerned with the question of whether a fixed-parameter tractable problem under consideration admits a small, and in particular a *polynomial*, kernel. For instance, the aforementioned Vertex Cover problem admits a polynomial kernel containing at most $2k$ vertices [86].

1.3.3 Parameterized Intractibility

As we have no proof of P $\neq$ NP, we cannot rule out the possibility that problems such as Clique are polynomial-time solvable and hence FPT. Therefore, our lower bound theory has to be conditional: we are proving statements of the form "if problem A has a certain type of algorithm, then problem B has a certain type of algorithm as well". If we have accepted as a working hypothesis that B has no such algorithms (or we have already proved that such an algorithm for B would contradict our working hypothesis), then this gives evidence that problem A does not have this kind of algorithms either. This leads us to the notion of *parameterized reduction*, which is roughly speaking a polynomial time algorithm that reduces an input of the problem B to an input of problem A, in a way that preserves the size of parameter. Using this notion of parameterized reduction, Downey and Fellows [64] introduced the so-called W-hierarchy in order to classify parameterized problems according to their hardness. Clique is an archetypical example of a W[1]-complete problem, and as such it does not admit an FPT algorithm unless $\mathsf{FPT} = \mathsf{W}[1]$.

The Exponential Time Hypothesis (ETH) is a conjecture stating that, roughly speaking, 3-SAT cannot be solved in subexponential running time in terms of the number of variables. ETH allows us to obtain stronger lower bounds. For example, we can prove results saying that (assuming ETH) a problem cannot be solved in time $2^{o(n)}$, or a parameterized problem cannot be solved in time $f(k)n^{o(k)}$, or a fixed-parameter tractable problem does not admit a $2^{o(k)}n^{\mathcal{O}(1)}$ time algorithm.

1.3.4 Decomposition Parameters

The most prominent example of a decomposition parameter is the treewidth of a graph, introduced by Robertson and Seymour while developing the Graph Minors Theorem [177]. Treewidth has now become one of the most frequently used tools in parameterized algorithms with many applications in a range of important fields of theoretical computer science, including artificial intelligence and parameterized complexity (see, e.g., [46, 190]). Intuitively, treewidth measures how well the structure of a graph can be captured by a tree-like structural decomposition. When the treewidth of a graph is small, or equivalently the graph admits a good *tree decomposition*, then many problems intractable on general graphs become efficiently solvable. Treewidth is widely used as a tool in various graph algorithms. For example, tree decompositions can be exploited to design fast dynamic programming algorithms which construct the solution from the leafs of the decomposition to the root. There is also a powerful *meta-theorem* by Courcelle [46] which establishes the tractability of decision problems definable in Monadic Second-Order logic on graphs of bounded treewidth. Given all the applications of treewidth, it is natural to ask if one could design other measures of structural complexity of graphs that would be algorithmically useful in cases where treewidth fails.

One particular decomposition parameter that measures a different form of structure is *rank-width* [168]. The motivation for rank-width comes from the observation that although treewidth is high for any dense graph, many computational problems are tractable on classes of graphs that are dense, but structured. An obvious example of such graphs are complete graphs, but one can also allow a little bit more freedom in the structure. Rank-width is related to the structural parameter clique-width, but supersedes it in various algorithmic aspects.

1.3.5 Distance to Triviality

Another widely used approach in the design of structural parameters is to measure the distance of the input from some known tractable class of inputs. The concept of *distance to triviality* has appeared independently under various names in many different fields of computer science, including Algorithmic Graph Theory, Reasoning, and Satisfiability. The term *modulator* has been coined in a general setting by Cai [36]. Here we mostly focus on the prevalent notion of modulators by vertex deletion. Specifically, for a fixed graph class $\mathcal{C}$ we say that a vertex set $X \subseteq V(G)$ of a graph G is a modulator to $\mathcal{C}$ if $G - X \in \mathcal{C}$. The problem of finding modulators to various graph classes and also using these to solve a number of problems has become a prominent research direction in the past years. In particular, modulators allow FPT algorithms to exploit the structure of graphs which are "near" a tractable graph class; the modulator size is often used as the parameter in such algorithms. For historical reasons, modulators often carry different names depending on the graph class $\mathcal{C}$, such as vertex cover, feedback vertex set or odd cycle transversal.

In the context of FPT algorithms on graphs, modulator-based algorithms are sometimes superseded by decomposition-based algorithms, specifically by algorithms which use treewidth or clique-width. As an example, graphs with small classical modulators to acyclic graphs, outerplanar graphs, or edgeless graphs always have small treewidth (on the other hand, this is not the case for, e.g., planar graphs or interval graphs). For this reason, classical modulators were used almost exclusively in the context of difficult problems which remain intractable on graphs of bounded treewidth [79, 83] and for kernelization, where standard width-based decompositions do not provide polynomial kernels [25].

1.4 Contributions

In this section, we describe the contributions of this thesis. We do not restrict our focus to only one specific problem; instead, we investigate the possibilities of exploiting both already established, but mainly new kinds of structure to obtain FPT algorithms and lower bounds for a variety of problems that emerge from various areas of computer science such as Algorithmic Graph Theory, Artificial Intelligence, or Optimization. In general, structural parameters can be divided into two groups based on the way they are designed. Namely, one can capture the structure of instances by decompositions or one can directly measure the distance to triviality via modulators.

This thesis is divided into three parts. The first two parts are dedicated to decomposition parameters and modulators. The third part then investigates how these two distinct approaches can be combined to obtain "hybrid" parameters that can capture forms of structure which are beyond the reach of a single approach.

1.4.1 Contributions towards Decomposition Parameters

Part I focuses on solving hard problems through the use of decomposition parameters. The first chapter, Chapter 4, in this part investigates possibilities of exploiting the well-known decomposition parameter treewidth to count the number of linear extensions of a poset. As the treewidth is a graph parameter, we need to work with a graph representation of the input posets. We study two different representations of these posets and obtain the following results:

- We provide the first evidence that the problem does not allow for an FPT algorithm parameterized by the treewidth of the cover graph (also called the Hasse diagram) unless $\mathsf{FPT} = \mathsf{W}[1]$. We remark that this complements the XP algorithm of Kangas et al. [131] and resolves an open problem recently posed in the Dagstuhl seminar on Exact Algorithms [125].
- We complement this negative result by obtaining an FPT algorithm for the problem when the parameter is the treewidth of the incomparability graph of the poset.

Afterwards in Chapter 5, we switch our focus and study the use of decomposition parameters for QBF. In spite of the close connection between QBF and SAT, many of the tools and techniques which work for SAT are not known to help for QBF, and this is especially true for decomposition-based techniques. Indeed, even though there are several FPT algorithms for SAT parameterized by well-known decomposition parameters such as treewidth, pathwidth, or rank-width [190, 94], the same is not true for QBF [15] under well-established complexity assumptions.

- We introduce a novel parameter *prefix pathwidth*, which is an extension of pathwidth that takes into account not only the structure of clauses in the formula, but also the structure contained in the quantification of variables.
- We show that we can use a prefix path decomposition of width bounded by the parameter k to solve the given QBF instance in FPT time.
- We develop new algorithmic techniques to obtain two distinct algorithms for computing prefix path decompositions—one polynomial-time approximation algorithm and an FPT algorithm.

1.4.2 Contributions towards Modulators

Part II is then devoted to investigating algorithmic applications of modulators for various problems. We start our study in Chapter 7 by investigating the problem of finding modulators (i.e., vertex deletion sets) to distance-hereditary graphs, a well-studied graph class which is particularly important in the context of vertex deletion due to its connection to the graph parameter rank-width.

- We present the first single-exponential FPT algorithm, specifically an algorithm running in time $\mathcal{O}(c^k \cdot n^{\mathcal{O}(1)})$ for input size n and some constant c, for vertex deletion to distance-hereditary graphs.
- We complement our result with matching asymptotic lower bounds based on the exponential time hypothesis.
- As an application of our algorithm, we show that a vertex deletion set to distance-hereditary graphs can be used as a parameter which allows single-exponential FPT algorithms for classical NP-hard problems.

The second problem we study is Directed Feedback Vertex Set (DFVS); this is the main topic of Chapter 8 . Once DFVS has been classified as fixed-parameter tractable by Chen et al. [42], one of the most natural follow-up question in parameterized complexity is “does it admit a polynomial kernel?”. We study the existence of a polynomial kernel for this problem parameterized by the size of the feedback vertex set (or equivalently a modulator to a forest) of the underlying undirected graph.

- We give a kernel with $\mathcal{O}(k^4)$ vertices for DFVS parameterized by the size of the feedback vertex set of the underlying undirected graph on general graphs.
- Moreover, we give a kernel with $\mathcal{O}(k)$ vertices for the same parameterized problem when the input digraph is embeddable on a surface of constant genus.

Finally in Chapter 9, we initiate the study of modulators for ILP by analyzing modulators which fracture the instance into small, easy-to-handle components.

- We introduce a novel parameter *fracture modulator*, which, if we represent the ILP instance by its so-called incidence graph, is basically the size of the modulator to components whose size is bounded by the parameter as well. Equivalently, one can think about this parameter as the number of global variables or global constraints in an otherwise “compact” instance.
- We identify and analyze three separate cases depending on whether we allow global variables only, global constraints only, or both.
- We obtain a near-complete complexity landscape for the considered parameters: in particular, we identify the circumstances under which they can be used to obtain FPT and XP algorithms.

1.4.3 Contributions towards Hybrid Parameters

Finally in Part III, we focus on expanding classical graph modulators towards new, more general classes. In particular, here we investigate what happens when a graph contains a modulator which is large but “well-structured” (in the sense of having bounded rank-width). Can such modulators still be exploited to obtain efficient algorithms? And is it even possible to find such modulators efficiently?

- We introduce a family of “hybrid” parameters that combine the decomposition and modulator approach.

Given a graph G and a fixed graph class $\mathcal{H}$, the new parameters capture (roughly speaking) the minimum number of “blocks”, which we call *split-modules*, of “small” rank-width of any modulator of G into $\mathcal{H}$. We call modulators with this structure *well-structured* *modulators*, and the minimum number of blocks in a well-structured modulator is then the *well-structure number* of G or $\mathrm{wsn}^{\mathcal{H}}(G)$.

In Chapter 11, we start by exploring the boundaries of this parameter for fixed-parameter tractability. Moreover, as with most structural parameters, virtually all algorithmic applications of the well-structure number rely on having access to an appropriate decomposition. Towards this goal we obtain following results:

- We develop an FPT algorithm for computing $\mathrm{wsn}^{\mathcal{H}}$ for any graph class $\mathcal{H}$ which can be characterized by a finite set of forbidden induced subgraphs.
- We design FPT algorithms for Vertex Cover and Clique parameterized by $\mathrm{wsn}^{\mathcal{H}}$.
- We develop a *meta-theorem* to obtain fixed-parameter algorithms for problems definable in Monadic Second Order (MSO) logic [50] parameterized by $\mathrm{wsn}^{\mathcal{H}}$.
- We show that, in general, solving MSO-Opt problems [50, 93] is not FPT when parameterized by $\mathrm{wsn}^{\mathcal{H}}$.

In the subsequent Chapter 12, we shift our attention towards obtaining kernels using the well-structure number as a parameter. Since $\mathrm{wsn}^{\mathcal{H}}$ lower bounds rank-width and rank-width is known not to admit polynomial kernels for nearly any NP-hard problems, one cannot hope to use $\mathrm{wsn}^{\mathcal{H}}$ for polynomial kernelization. Hence, to obtain kernels, we need to restrict our parameter a little bit—instead of allowing the rank-width of split modules to be bounded by a function of the parameter, we bound it by some constant c. We call this the c-well-structure number of a graph G or $\mathrm{wsn}_c^{\mathcal{H}}(G)$. Using this notion, we obtain the following results:

- We develop a polynomial time approximation algorithms for computing $\mathrm{wsn}_c^{\mathcal{H}}$ to a range of graph classes.
- We show that whenever a modulator to a graph class $\mathcal{H}$ can be used to poly-kernelize some MSO-definable problem, this problem also admits a polynomial kernel when parameterized by the c-well-structure number for $\mathcal{H}$ as long as c-well-structured modulators to $\mathcal{H}$ can be approximated in polynomial time.
- The remainder of Chapter 12 then deals with specific applications of these results.

Roadmap

After this introductory chapter, in Chapter 2, we provide an overview of relevant concepts and results from the areas of graph theory, classical complexity theory, and parameterized complexity theory, respectively. The remainder of this thesis, after the preliminaries chapter, is divided into four parts. The first three parts are devoted towards obtaining our main results discussed in the previous section and in the last part we summarize once again our contributions and propose some questions for the future investigation.

Notes

The results in this thesis appeared in conference papers in the proceedings of WADS 2015 [74], IPEC 2015 [73], MFCS 2016 [71], ESA 2016 [70] AAAI 2016 [72], MFCS 2017 [21] IJCAI 2017 [67], as well as in a journal paper in Algorithmica [75].

CHAPTER 2

Preliminaries

In this chapter, we give concise introduction to the terminology and notation that we use throughout the thesis.

For a set A, we denote by $[A]^\ell$ the set of all ℓ-elements subsets of A. The set of all integers is denoted by $\mathbb{Z}$, the set of nonnegative integers by $\mathbb{N}_0$, and the set of natural numbers (that is, positive integers) by $\mathbb{N}$. For integers n, m with $n \leq m$, we let $[n, m] := \{n, n+1, \ldots, m\}$ and $[n] := [1, n]$. Unless mentioned explicitly otherwise, we encode integers in binary.

Now we continue by introducing standard graph theoretic notions. Most of the terminology and notation introduced in this Chapter is the same or similar to the book by Diestel [59].

2.1 Graph Theory

A graph is a pair $G = (V, E)$, where V is a finite set, whose elements are referred to as *vertices* of the graph G and $E \subseteq [V]^2$ is a set of 2-element subsets of V and its elements are referred to as *edges*. The usual way to picture a graph is by drawing a dot for each vertex and joining two of these dots by a line if the corresponding two vertices form an edge. Just how these dots and lines are drawn is considered irrelevant: all that matters is the information which pairs of vertices form an edge and which do not.

The vertex and edge set of a graph G are usually denoted by $V(G)$ and $E(G)$, respectively. These conventions are independent of any actual names of these two sets: the vertex set W of a graph $H = (W, F)$ is still referred to as $V(H)$. We do not always strictly distinguish between a graph and its vertex or edge set. For example, we may say that a vertex v is in G instead of $V(G)$.

A vertex v is incident with an edge e, if $v \in e$; then e is a edge at v. The two vertices incident with an edge are its *endvertices*, or sometimes *endpoints*, or simply *ends*.

An edge $\{x, y\}$ is usually written as xy (or yx). We also sometimes say that xy is an edge between vertices x and y. Two vertices x and y of G are adjacent, or neighbors, if xy is an edge of G. The *(open) neighborhood* of a vertex x in G is the set of all neighbors of x in G, or more formally the set $\{y \in V(G) : xy \in E(G)\}$, and is denoted by $N_G(x)$. The *closed neighborhood* $N_G[x]$ of x is defined as $N_G(x) \cup \{x\}$. If the graph G is clear from the contex, we often drop the subscript G. The *degree* $d_G(v) = d(v)$ of a vertex v is the number of edges at v, which is equal (by the definition of the graph we are using in this thesis) to the number of neighbors of v. The edges $e \neq f$ are *adjacent*, if they have a common endvertex. If all the vertices of G are pairwise adjacent, the G is *complete*. A set of vertices is independent, if no two of its elements are adjacent.

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. If $V' \subseteq V$ and $E' \subseteq E$, then G' is a subgraph of G (G is a supergraph of G'), written as $G' \subseteq G$. In this case, we also say that G *contains* G'. If $G' \subseteq G$, and G' contains precisely all the edges $xy \in E$ with $x, y \in V'$, then G' is an induced subgraph of G; we say V' *induces* the graph G' in G, and write $G' = G[V']$. For a subgraph H of G, if a vertex v is adjacent to some vertex $u \in V(H)$ in G, we sometimes say that u is adjacent to H or u is adjacent to $V(H)$. If U is a set of vertices of G, we write $G - U$ (or $G \setminus U$) for $G[V(G) \setminus U]$. If $U = \{v\}$ is a singleton, we write $G - v$ rather then $G - \{v\}$. For a subset $F \subseteq [V]^2$, we write $G - F = (V, E \setminus F)$ and $G + F = (V, E \cup F)$; as above, $G - \{e\}$ and $G + \{e\}$ are abbreviated to $G - e$, $G + e$. The complement $\bar{G}$ of G is the graph $(V, [V]^2 \setminus E)$.

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. We call G and G' isomorphic, and write $G \simeq G'$, if there exists a bijection $\phi : V \to V'$ with $xy \in E \Leftrightarrow \phi(x)\phi(y) \in E'$ for all $x, y \in V$. Such a map ϕ is called an isomorphism; if $G = G'$, it is called an automorphism. We do not normally distinguish between isomorphic graphs. Thus, we usually write $G = G'$ rather than $G \simeq G$, speak of the complete graph on 17 vertices, and so on.

2.1.1 Paths and Cycles

A path is a graph $P = (V, E)$ of the form

$$V = \{x_0, x_1, \dots, x_\ell\} \text{ and } E = \{x_0x_1, x_1x_2, \dots, x_{\ell-1}x_\ell\}.$$

The vertices x_0 and x_ℓ are *linked* by P and are called its *ends*; the vertices $x_1, \dots, x_{\ell-1}$ are the *inner (or internal)* vertices of P. The number of edges of a path is its *length*. We often refer to a path by the natural sequence of its vertices, writing, $P = x_0x_1 \dots x_\ell$ and calling P a path from x_0 to x_ℓ or a path between x_0 and x_ℓ. We denote by P_n the path on n vertices. We say that a path P is *in* G, if $P \subseteq G$. A path P in G is an induced path if $P = G[V(P)]$, or in other words there is no edge e in G between two vertices in P such that $e \notin E(P)$. Given sets A, B of vertices, *A-B path* is a path from a vertex in A to a vertex in B with all internal vertices disjoint from $A \cup B$.

If $P = x_0 \dots x_\ell$ is a path and $\ell \geq 2$, then the graph $C := P + x_\ell x_0$ is called a *cycle*. Similarly as with paths, we often denote a cycle by its (cyclic) sequence of the vertices;

the above cycle C could be written as $x_0 \dots x_\ell x_0$. The length of a cycle is the number of its edges and we denote by C_ℓ the cycle with the length ℓ.

The *distance* $d_G(x, y) = d(x, y)$ in G of two vertices x, y is the length of a shortest x-y path in G; if no such path exists we set $d(x, y) = \infty$.

2.1.2 Complete Graph

A graph is called *complete* if every pair of vertices is adjacent. If two complete graphs have the same number of vertices, then they are isomorphic. The standard model for a complete graph on n vertices is denoted K_n.

2.1.3 Connectivity

A non-empty graph G is called *connected* if every two of its vertices are linked by a path in G. A maximal connected subgraph of G is called a *(connected) component* of G. We denote by $\mathrm{cc}(G)$ the number of connected components of G. If $A, B \subseteq V(G)$ and $X \subseteq V(G) \cup E(G)$ are such that every A-B path contains a vertex or an edge in X, we say that X *separates* the sets A and B in G, or equivalently X is a A-B *separator* in G. If A or B contains a single vertex x, then we often write x instead of $\{x\}$. We say that X is a *separator in* G if there are two vertices $a, b \in V(G) \setminus X$ such that X is a a-b separator. If $X \subseteq V(G)$ or $X \subseteq E(G)$, then we say that X is a vertex separator or an edge separator in G, respectively. An edge e of a connected graph G is a cut edge if the graph obtained from G by removing e is disconnected. Similarly, a vertex v of a connected graph G is a cut vertex if the graph obtained from G by removing v is disconnected.

A graph $G = (V, E)$ is said to be k-*connected* (for $k \in \mathbb{N}$) if $|V| > k$ and $G - X$ is connected for every set $X \subseteq V$ such that $|X| < k$.

2.1.4 Trees and Forests

A graph that does not contain a cycle as a subgraph is *acyclic*. An acyclic graph is also called a forest. A connected forest is called a tree. Thus a forest is a graph whose components are trees. A star is a tree with a distinguished vertex, called the *center*, adjacent to all other vertices. The vertices of degree 1 are called *leaves*. Note that every non-trivial tree has at least 2 leaves and if we remove a leave from a tree, then the remaining graph is still a tree. The non-leaf vertices of a tree are called its *internal nodes*. We mention here few important properties of a tree that follows straightforwardly from the definition of tree.

Fact 2.1 ([59]). *The following assertions are equivalent for a graph T:*

- *T is a tree;*
- *every two vertices of a tree T are linked by a unique path;*
- *T is connected and has $n - 1$ edges.*

Sometimes it is convenient to consider one vertex of a tree as special; such a vertex is then called the *root* of this tree. A tree with a fixed root is a *rooted tree*. Note that choosing a root r in tree T imposes a partial ordering on $V(T)$. For a vertex v, the vertices of the unique v-r path other then v are called *ancestors* of v. The neighbor of v in this path is a *parent* vertex of v. If u is the parent of v, then v is said to be a *child* of u.

2.1.5 Bipartite graphs

let $r \geq 2$ be an integer. A graph $G = (V, E)$ is called r*-partite* if V admits a partition into r classes such that every edge in E has its endvertices in different partitions. Instead of 2-partite one usually says *bipartite*. Clearly, bipartite graphs cannot contain an odd cycle.

Fact 2.2. *A graph is bipartite if and only if it does not contain an odd cycle.*

An r-partite graph in which every two vertices from different partition classes are adjacent is called *complete*; the complete r-partite graphs for all r together are the *complete multipartite* graphs. The complete r-partite graph with the partition sizes $n_1, \ldots, n_r$, respectively, is denoted by $K_{n_1,\ldots,n_r}$.

2.1.6 Contraction and Minors

Let $e = xy$ be an edge of a graph $G = (V, E)$. By G/e we denote the graph obtained from G by *contracting* the edge e into a new vertex v_e, which becomes adjacent to all the former neighbors of x and of y. Formally, G/e is a graph (V_0, E_0) with vertex set $V_0 := (V \setminus \{x, y\}) \cup \{v_e\}$ (where v_e is the 'new' vertex, i.e., $v_e \notin V \cup E$) and edge set $E_0 := \{vw \in E \mid \{v, w\} \cap \{x, y\} = \emptyset\} \cup \{v_e w \mid xw \in E \setminus \{e\} \text{ or } yw \in E \setminus \{e\}\}$. A graph H is said to be a *minor* of another graph G, if a graph isomorphic to H can be obtained from G by contracting some edges, deleting some edges, and deleting some vertices. The order in which a sequence of such contractions and deletions is performed on G does not affect the resulting graph H. We say that a graph G is H*-minor-free* if G has no minor isomorphic to H.

Fact 2.3. *The minor relation is a partial ordering on the class of finite graphs, i.e., it is reflexive, antisymmetric and transitive.*

2.1.7 Other notions of graphs

We mention here some other notions of graphs which are mentioned or used in certain parts of the thesis.

A multigraph is a pair (V, E), where V is a set of vertices and E is a multiset (each element can appear multiple times) containing elements of $V \cup [V]^2$. Note that a multigraph may have several edges between the same two vertices x, y. Such edges are called multiple edges. To express that x and y are the ends of an edge e we still write $e = xy$, though this no longer determines e uniquely. The edges containing only one vertex are called loops. A graph is thus essentially the same as a multigraph without loops or multiple edges.

A *directed graph (or simply digraph)* is a pair (V, E) of disjoint sets, where $E \subseteq V \times V$. We call the set V the vertex set and its elements are vertices. The set E is the arc set and its elements are called arcs or directed edges. Similarly, the definition of directed graph can be extended to a *directed multigraph*, where we allow multiple edges between the same set of vertices. Moreover, if these edges have the same direction (say from x to y), they are *parallel*. For a directed edge $e = (x, y)$ we call the vertex x the tail of the edge e and y the head of the edged e.

Most of the notions defined in this section for graphs extend straightforwardly to multigraphs and digraphs. For a detailed treatment of multigraphs and directed graphs, the reader is referred to books by Diestel [59] or Bang-Jensen and Gutin [18]. We introduce here some notions that differ from the standard graph and will be useful throughout the thesis.

In the following let D be a directed (multi)graph. We say that a vertex u is a *neighbor* of a vertex v in D if at least one of $(u, v) \in E(D)$ or $(v, u) \in E(D)$ holds. We denote by $N_D(v)$ (or by $N(v)$ if D is clear from the context) the set of all neighbors of v in D. A vertex u is an *in-neighbor* or *out-neighbor* of a vertex v in D if $(v, u) \in E(D)$ or $(u, v) \in E(D)$, respectively. We denote by $N_D^-(u)$ and $N_D^+(u)$ the set of all in-neighbors and out-neighbors of v in D, respectively. Again we drop the subscript D if it can be inferred from the context and for a vertex set $A \subseteq V(D)$ we write $N_D^-(A)$ and $N_D^+(A)$ to denote the sets $(\bigcup_{a\in A} N_D^-(a)) \setminus A$ and $(\bigcup_{a\in A} N_D^+(a)) \setminus A$, respectively. The *in-degree* $d_D^-(v) = d^-(v)$ of vertex v is the number of edges with v being the head of the edge. The *out-degree* $d_D^+(v) = d^+(v)$ of vertex v is the number of edges with v being the tail of the edge. The *total degree* $d_D(v) = d(v)$ of vertex v is the sum of its in-degree and its out-degree. Note that loops are counted twice for the total degree, so if there is only the arc (v, v) incident to v then its total degree is 2.

A directed path or simply *path* P in D from a vertex u to a vertex v is a sequence $(v_0, \dots, v_l)$ such that $v_0 = u$, $v_n = v_l$, $(v_i, v_{i+1}) \in E(D)$ for every i with $0 \leq i < l$, and apart from the pair (v_0, v_l) all pairs of vertices in P are disjoint. We call v_0 and v_l the *endpoints* of P and the vertices $v_1, \dots, v_{l-1}$ the *internal vertices* of P. For two vertex sets A and B we say that there is a directed path from A to B if there are $a \in A$ and $b \in B$ such that D has a directed path from a to b. We say that a set of paths is *vertex disjoint* if no two paths in the set share an internal vertex.

In Chapter 8 the following definition of edge-contraction in directed graphs will be useful. Given an arc $(u, v) \in E(D)$, we say that the directed graph D' is obtained from D after *contracting* the arc (u, v) if D' is obtained from D after replacing u and v in D with a new vertex n and adding all arcs (w, n) for every arc (w, u) or (w, v) in D as well as all arcs (n, w) for every arc (u, w) or (v, w) in D. We note that several notions of contraction exist for directed graphs such as for example *butterfly contraction* [130].

Finally, a hypergraph is a pair (V, E) of disjoint sets, where the elements of E are non-empty subsets (of any cardinality) of V. Thus, graphs are a special case of hypergraphs.

2.2 Classical Complexity

In general we study the complexity of computing a function whose input and output are finite strings over some fixed alphabet Σ (for example binary strings, i.e., $\Sigma = \{0, 1\}$). We identify these functions with computational problems. More precisely, given a function f, we define the computational problem Q_f to be the question to compute the value $f(x)$ for a given input x (we will use the terms *input* and instance interchangeably). Note that simple encodings can be used to represent general mathematical objects – integers, pairs of integers, graphs, vectors, matrices, etc. – as strings. For example, we can represent a graph as its adjacency matrix (i.e., for an n vertex graph $G = (V, E)$, choose an arbitrary ordering of vertices in V, given by a bijection $\varphi : [n - 1] \to V$, and represent G by an $n \times n$ 0/1-valued matrix A such that $A_{i,j} = 1$ if and only if the edge $\varphi(i)\varphi(j)$ is present in G). However, we will avoid dealing explicitly with such low level issues of representation, and will alway assume that we have some canonical (and unspecified) string representation of the object.

We will work with several special cases of computational problems in this thesis. An important special case of functions mapping strings to strings is the case of functions whose output is a single bit. We identify such a function f with the set $Q_f = \{x \mid f(x) = 1\}$ and call such sets *languages* or *decision problems*. In this setting, the computational problem of computing f (i.e., given x, compute $f(x)$) is equivalent to deciding whether $x \in Q_f$. If $x \in Q_f$, we say that x is an Yes-*instances* of Q_f, otherwise we say that x is a No-*instance*. A search problem is a computational problem which is associated with a binary relation R instead of a function. To solve a search problem for input x it suffices to output an arbitrary y such that $(x, y) \in R$. A counting problem asks for the number of solutions to a given search problem, i.e., given a binary relation R, the counting problem associated with R is the function $f_R(x) = |\{y : (x, y) \in R\}|$. An optimization problem asks for finding a "best possible" solution among the set of all possible solutions to a search problem. One example is the Vertex Cover problem: "Given a graph G, find a vertex cover of G of minimum size."

Given a problem Q_f, we say that algorithm $\mathcal{A}$ *solves* Q if for all inputs $x \in \Sigma^*$ the algorithm $\mathcal{A}$ outputs $f(x)$. Complexity theory studies the number of steps required by an optimal algorithm to solve such problems. Here, the number of steps is measured in terms of the input size $|x| = n$. In most cases, a worst-case perspective is taken. This means that for each input size n, we measure the maximum number of steps that the algorithm takes on any input of size n. When expressing the running time of an algorithm, we focus on upper bound guarantees. For instance, we say that an algorithm runs in time n^2 if for each input $x \in \Sigma^*$, the algorithm takes at most (but possibly less than) $|x|^2$ steps.

Moreover, when we express the running time of algorithms, we will often be more interested in the *rate of growth* of functions upper bounding the running time than their precise behavior. In order to explain this more precisely, we introduce "Big Oh" and "little oh" notation. We also overview several related notions.

The Big Oh and little oh notations suppress unimportant details and allow us to focus on salient features of running times. Let $f, g : \mathbb{N} \to \mathbb{N}$ be two arbitrary functions. Then we say that f is of the *order of* g, $f = \mathcal{O}(g)$ or $f(n) = \mathcal{O}(g(n))$, if there are constants $c, n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. Intuitively, if $f(n) = \mathcal{O}(g(n))$, then $f(n)$ *grows asymptotically at most as fast as* $g(n)$. Moreover, if for every $\epsilon > 0$ there exists $n_0 \in \mathbb{N}$ such that $f(n) \leq \epsilon \cdot g(n)$ for all $n \geq n_0$, we say that $f(n) = o(g(n))$. Intuitively, if $f(n) = o(g(n))$, it means that $g(n)$ *grows asymptotically faster* than $f(n)$. There also exist counterparts to Big Oh and little oh that express lower bounds rather than upper bounds. In particular, we have $f(n) = \Omega(g(n))$ if $g(n) = \mathcal{O}(f(n))$ and $f(n) = \omega(g(n))$ if $g(n) = o(f(n))$.

2.2.1 Turing Machines

To formally define what we mean by an algorithm and the running time of an algorithm, we need to utilize the notion of Turing machines, which were introduced in the foundational work of Alan Turing [195]. Roughly speaking a Turing machine is a machine with an infinite tape (Turing machines with multiple tapes are also commonly considered) and one head that can read from the tape, write on the tape, and either move on the tape one cell left or right or stay at the position of the current cell. At the beginning we assume that the head is on the first letter of the input word and all the cells not containing letters of the input word are blank, which is marked by a special symbol $\mathbf{B}$.

More formally, a non-deterministic Turing machine is a tuple $M = (S, \Sigma, \Gamma, \delta, s_0, F)$, where:

- S is the finite, non-empty set of *states*;
- Γ is the finite, non-empty *tape alphabet*;
- $\Gamma \cap S = \emptyset$;
- $\Sigma \subseteq \Gamma$ is the finite, non-empty *input alphabet*;
- $s_0 \in S$ is the *initial state*;
- $F \subseteq S$ is the set of accepting states; and
- $\delta : S \times (\Gamma \cup \{\mathbf{B}\}) \to 2^{S \times \Gamma \times \{-1,0,1\}}$ is the *transition function*.

Here $\mathbf{B} \notin \Gamma \cup S$ is a special *blank* symbol. A *configuration* is a tuple (s, x, p), where $s \in S$, $x \in \Gamma^*$, and $p \in [0, |x| + 1]$. Intuitively, x is the word currently on tape, disregarding blanks, s is the state of the machine and p is the position of the head. Since the head is not allowed to write down $\mathbf{B}$, the head has to be either on a position somewhere inside x or on the blank just before or just after x. The *initial configuration* for an input $x \in \Sigma^*$ is $C_0(x) = (s_0, x, 1)$.

A *computation step* is a pair of configurations (C_1, C_2) such that if $C_1 = (s_1, xay, p_1)$, $x, y \in \Gamma^*$, $a \in \Gamma \cup \mathbf{B}$ and $|x| = p_1 - 1$, then $C_2 = (s_2, xby, p_2)$ and $(s_2, b, p_2 - p_1) \in \delta(s_1, a)$.

If (C_1, C_2) is a computation step of a Turing machine $\mathbb{M}$, we call C_2 a *successor configuration* of C_1. A *halting configuration* is a configuration that has no successor configuration. A halting configuration is *accepting* if its state is in F. We say that the machine $\mathbb{M}$ accepts an input x if there exists a sequence of configurations $C_0, C_1, \ldots, C_n$ such that $C_0 = C_0(x)$ is the initial configuration and (C_{i-1}, C_i) is a computation step for each $i \in [n]$. We call such a sequence a *computation*. We identify *non-deterministic algorithms* with non-deterministic Turing machines and the length of a shortest computation, if there is one, ending with an accepting halting configuration is then the *running time* of the algorithm on the input x. For a decision problem Q, we then say that Turing machine $\mathbb{M}$ *solves* Q, if for each $x \in Q$ the Turing machine $\mathbb{M}$ accepts x. We say that the running time of $\mathbb{M}$ if $f(n)$ if for each $x \in Q$ such that $|x| = n$, the running time of $\mathbb{M}$ on x is at most $f(|x|)$. Similarly, for a function $f : \Sigma^* \to \Sigma^*$, we say that $\mathbb{M}$ *solves* f (or equivalently the problem Q_f associated with f), if for every $x \in \Sigma^*$ there is a computation that ends in an accepting halting configuration $C = (s, y, p)$, where the restriction of y to Σ is exactly $f(x)$.

A *deterministic* Turing machine machine is a Turing machine with $|\delta(s, x)| \leq 1$ for all $s \in S$ and all $x \in \Gamma \cup \{\mathbf{B}\}$. Note that in this case, each configuration C has at most one possible successor configuration. Deterministic algorithm and running time for a problem Q are then defined similarly to non-deterministic ones with a distinction that the algorithm is required to end in a non-accepting halting configuration for all $x \notin Q$ and the running time of the deterministic Turing machine $\mathbb{M}$ is $f(n)$ if for every $x \in \Sigma^*$ such that $|x| = n$ the unique computation starting in the initial configuration for x, $C_0(x)$, and ending in a halting configuration has length at most $f(n)$. If we do not specify otherwise, by an algorithm we always mean a deterministic algorithm.

2.2.2 Classical Complexity: P vs NP

Traditionally, the notions of tractability and tractable problems were identified with the complexity class P. This complexity class contains precisely the decision problems that have algorithms (deterministic Turing machines) with running time polynomial in the input size ($\mathcal{O}(n^c)$ for some constant c).

Using the notion of tractability, it is easy to define intractability: a problem is intractable if it is not tractable. However, it turned out that this naive notion of intractability is unproductive for a large class of problems from many areas of computer science. For these problems, nobody has been able to find a polynomial-time algorithm, and nobody has been able to prove that no such algorithm exists. For this reason, the concept of NP-completeness was introduced. The complexity class NP consists of all decision problems that are solvable in polynomial time using a non-deterministic Turing machine. There are problems in the class NP for which the best known (deterministic) algorithms run in time $2^{\Omega(n)}$. To substantiate the suspicion that a certain problem is not polynomial-time solvable, one can relate this problem to other problems in NP using the concept of reductions.

Let Q_1 and Q_2 be two decision problems. A polynomial-time reduction from Q_1 to Q_2 is a polynomial-time algorithm that for each input $x_1 \in \Sigma^*$ produces an output x_2 such that $x_1 \in Q_1$ if and only if $x_2 \in Q_2$. (Such reductions are also called many-to-one reductions, or Karp reductions.) We then say that a problem Q is NP-hard if for each problem $Q_0 \in \mathsf{NP}$, there is a polynomial-time reduction from Q_0 to Q. Intuitively, an NP-hard problem Q is as hard as any other problem in NP, because if Q were polynomial-time solvable, then each problem in NP would be polynomial-time solvable. A problem Q is NP-complete if it is both in NP and NP-hard. A practically useful way of proving NP-completeness is offered by the Cook-Levin Theorem [45, 147]. This seminal result identified a first NP-complete problem: SAT. As a result, subsequent NP-hardness proofs only need to provide a polynomial-time reduction from this single problem, rather than providing that there is an reduction from all problems in NP.

There also exists another notion of reduction for computational problems. A Turing reduction from a problem Q_1 to Q_2 is a polynomial algorithm $\mathbb{A}$ solving Q_1, where $\mathbb{A}$ is a deterministic Turing machine with an oracle for Q_2. Such oracle Turing machine in addition to the work tape and the read/write work head has also an *oracle tape*, an *oracle head* and two special states ASK and RESPONSE. The oracle Turing machine is allowed to go from a configuration in the state ASK to the state RESPONSE, in a single computation step, replacing the non-blank part x of the oracle tape by the output of Q_2 on input x and setting the oracle head to the first non-blank on the oracle tape.

2.2.3 List of NP-complete Problems

Throughout the thesis we deal with several well-studied NP-complete problems, here we list some of them.

Problem 1: SAT

In our setting, a propositional formula is constructed from individual binary variables using usual Boolean connectives $(\neg, \wedge, \vee, \rightarrow, \leftrightarrow)$.

Instance: A propositional formula φ.
Question: Can we assign truth values to the variables of φ such that formula evaluates to true?

Problem 2: d-CNF-SAT, for $d \geq 3$.

For a set of propositional variables K, a literal is either a variable $x \in K$ or its negation $\neg x$, where $v(x) = v(\neg x) = x$ denotes the variable of a literal. A clause is a disjunction over literals. A *propositional formula in conjunctive normal form* (i.e., a CNF formula) is a conjunction over clauses. We say that a CNF formula ϕ is *over* a variable set K if each literal x in ϕ satisfies $v(x) \in K$, and denote the set of variables which occur in ϕ by $\text{var}(\phi)$. For notational purposes, we will view a clause as a set of literals and a CNF formula as a set of clauses.

> *Instance*: A CNF formula φ such that each clause of φ contains at most d literals.
> *Question*: Can we assign truth values to the variables of φ such that formula evaluates to true?

Problem 3: VERTEX COVER

> *Instance*: A graph G and an integer m.
> *Question*: Is there a set $S \subseteq V(G)$ of cardinality at most m such that $G \setminus S$ does not contain any edge?

Problem 4: (UNDIRECTED) FEEDBACK VERTEX SET ((U)FVS)

> *Instance*: A graph G and an integer m.
> *Question*: Is there a set $S \subseteq V(G)$ of cardinality at most m such that $G \setminus S$ is forest?

Problem 5: DIRECTED FEEDBACK VERTEX SET (DFVS)

> *Instance*: A digraph D and an integer m.
> *Question*: Is there a set $S \subseteq V(D)$ of cardinality at most m such that $G \setminus D$ is acyclic?

Problem 6: INDEPENDENT SET

> *Instance*: A graph G and an integer m.
> *Question*: Is there an independent set in G of cardinality at least m?

Problem 7: CLIQUE

> *Instance*: A graph G and an integer m.
> *Question*: Is there a clique in G of cardinality at least m?

Problem 8: Dominating Set

> *Instance*: A graph G and an integer m.
> *Question*: Is there a set $S \subseteq V(G)$ of cardinality at least m such that each vertex in G is either in S or has a neighbor in S?

Problem 9: Hitting Set

> *Instance*: A ground set U and a collection $\mathcal{C}$ of subsets of S, and integer m.
> *Question*: Is there a set $S \subseteq U$ of cardinality at most m, which intersects each set in $\mathcal{C}$?

Problem 10: d-Hitting Set

> *Instance*: A ground set U and a collection $\mathcal{C}$ of subsets of S, each of cardinality at most d, and integer m.
> *Question*: Is there a set $S \subseteq U$ of cardinality at most m, which intersects each set in $\mathcal{C}$?

Problem 11: Equitable Coloring

> *Instance*: A graph G and an integer r.
> *Question*: Does G admit a proper r-coloring such that the number of vertices in any two color classes differ by at most one?

We note that, by definition, problems in NP are decision problems. However, as we can see from the above examples, many of them ask if there exists some certificate (such as a set of m vertices forming a clique). Therefore, throughout the thesis, we often consider the search or the optimization variant of these problems under the same name. It is known that both of these variants of a decision problem admit a polynomial algorithm if and only if the decision problem does. Hence, considering the tractability of problems, this does not cause any obstacles. For a further discussion about decision vs search problems see for example the book by Arora and Barak [13] or any other book or undergraduate text dealing with computational complexity.

2.2.4 Exponential Time Hypothesis

SAT is an archetypal NP-complete problem and it is commonly believed that there is no algorithm solving SAT in time $2^{o(n)}$. This assumption has been formalized by Impagliazzo and Paturi as an Exponential Time Hypothesis [126].

Definition 2.4 (Exponential Time Hypothesis (ETH) [127])**.** There exists a constant $s > 0$ such that 3-CNF-SAT with n variables and m clauses cannot be solved in time $2^{sn}(n+m)^{\mathcal{O}(1)}$.

We will use the fact that the classical Vertex Cover problem cannot be solved in subexponential time under ETH.

Theorem 2.5 (Cai and Juedes [38]). *There is no $2^{o(k)} \cdot |V(G)|^{\mathcal{O}(1)}$ algorithm for* Vertex Cover*, unless ETH fails.*

2.2.5 Polynomial Space

Next, we briefly consider the complexity class PSPACE , consisting of all decision problems that can be solved by an algorithm that uses a polynomial amount of space (or memory). The amount of *space* that a Turing machine uses for an input $x \in \Sigma^*$ is defined as the number of tape cells to which it writes during the computation. In other words, the class PSPACE consists of all problems that can be solved using space $\mathcal{O}(n^c)$, for some constant $c \in \mathbb{N}$. As a consequence of Savitch's Theorem [184], it is well-known fact that NP $\subseteq$ PSPACE.

2.2.6 Monadic Second Order Logic

Here we introduce monadic second order logic (MSO), which will play a crucial role throughout the thesis. Roughly speaking, MSO is the fragment of second-order logic where the second-order quantification is limited to quantification over sets. We refer to the books by Courcelle and Engelfriet [48] or Libkin [148] for a more detailed foray into monadic second order logic and related notions.

We assume that we have an infinite supply of individual variables, denoted by lowercase letters x, y, z, and an infinite supply of set variables, denoted by uppercase letters X, Y, Z. *Formulas* of MSO are built up from atomic formulas using the usual Boolean connectives $(\neg, \wedge, \vee, \rightarrow, \leftrightarrow)$, quantification over individual variables ($\forall x$, $\exists x$), and quantification over set variables ($\forall X$, $\exists X$).

When speaking about MSO over graphs, we in general distinguish two types of MSO, notable MSO_1 and MSO_2.

MSO_1 Individual variables range over vertices, and set variables range over sets of vertices. The atomic formula Exy expresses adjacency, $x = y$ expresses equality, and Xx expresses that the vertex x is in the set X.

MSO_2 Individual variables range over vertices or edges, and set variables range over sets of vertices or edges. The atomic formula Ixy expresses that the vertex x is incident with the edge y, $x = y$ expresses equality, and Xx expresses that the vertex or the edge x is in the set X.

Sometimes we also speak about MSO over labeled graphs, in which we assume that the logic (MSO_1 or MSO_2) contains also an atomic formula $P_a x$ that expresses that "vertex or edge x has label a". Moreover, MSO_2 over directed graphs contains atomic formulas Hxy

expressing “vertex x is the head of the edge y” and Txy expressing “vertex x is the tail of the edge y”. We remark here that MSO_2 logic is more general of the two. In particular, all formulas from MSO_1 can be easily translated to equivalent formula in MSO_2 by simply replacing atomic formulas Exy by $\exists e(Ixe \wedge Iye)$. However, some properties expressible in MSO_2 cannot be expressed in MSO_1. An example of such property is that a simple graph has a Hamiltonian cycle (see for example Proposition 5.13 in the book by Courcelle and Engelfriet [48]).

Free and bound variables of a formula are defined in the usual way. A sentence is a formula without free variables. We write $\varphi(X_1, \dots, X_n)$ to indicate that the set of free variables of formula φ is $\{X_1, \dots, X_n\}$. If $G = (V, E)$ is a graph and $S_1, \dots, S_n \subseteq V$ we write $G \models \varphi(S_1, \dots, S_n)$ to denote that φ holds in G if the variables X_i are interpreted by the sets S_i, for $i \in [n]$. For a fixed MSO sentence φ, the problem framework we are mainly interested in is formalized below.

monadic second order logic!Model Checking2@MSO-MC$_\varphi$ MSO Model Checking (MSO-MC$_\varphi$)
Instance: A graph G.
Question: Does $G \models \varphi$ hold?

While MSO model checking problems already capture numerous graph problems (e.g., 3-Coloring), there are some well-known problems on graphs that cannot be captured in this way, such as Vertex Cover, Dominating Set, and Clique. Many such problems can be formulated in the form of *MSO optimization problems*. Let $\varphi = \varphi(X)$ be an MSO formula with one free set variable X and $\diamondsuit \in \{\leq, \geq\}$.

MSO-Opt$_\varphi^\diamondsuit$
Instance: A graph G and an integer $r \in \mathbb{N}$.
Question: Is there a set $S \subseteq V(G)$ such that $G \models \varphi(S)$ and $|S| \diamondsuit r$?

2.3 Parameterized Complexity

Parameterized complexity was first developed by Downey and Fellows, and expanded in a series of papers [1, 37, 60, 61, 65] culminating in a monograph by Downey and Fellow [63]. The area has since greatly expanded. In this section, we give a brief overview of the area and the notions that are crucial for our thesis. For more detailed treatment of parameterized complexity, we refer reader to several books on the topic by Downey and Fellow [63, 64], Cygan et al. [53], Flum and Grohe [86], Niedermeier [162], or a habilitation thesis by Fernau [82].

2.3.1 Fixed-Parameter Tractability

The main idea behind fixed-parameter tractability is the observation that the worst-case classically taken in the complexity theory measure the complexity of the problem only by the size of the input. However, in many cases you know more about the input than just its size. Parameterized complexity provides tools for a multi-dimensional complexity analysis where you can take such additional information about the input into account.

The concept of identifying information about problem inputs that can be exploited algorithmically is formally captured in parameterized complexity by the notions of parameterized problems and fixed-parameter tractability. A parameterized problem Q is a function $q : \Sigma^* \times \mathbb{N} \to \Sigma^*$. For an instance (x, k) of Q, we say that x is the *main part* of the instance and k the parameter. A parameterized problem Q is fixed-parameter tractable if there exists a computable function $f : \mathbb{N} \to \mathbb{N}$, a constant $c \in \mathbb{N}$ and and algorithm that on input (x, k) runs in time $f(k) \cdot |x|^c$ and outputs $q(x, k)$. Such an algorithm is called a *fixed-parameter* algorithm or shortly an *FPT* algorithm and such running time is called *fixed-parameter* or FPT time. This definition of parameterized problem straightforwardly generalizes also to parameterized decision/search/counting/optimization problems. The class of all *parameterized decision problems* that are fixed-parameter tractable is denoted by FPT. For a classical problem P, we use $P[k]$ to denote the parameterized problem P parameterized by the parameter k.

2.3.2 Polynomial Time Solvability for a Fixed Parameter and the Class XP

Even though the name "fixed-parameter tractable" hints that the concept of fixed-parameter tractability can be described by the fact that for every fixed constant value of the parameter the problem is polynomial time tractable, it is not true in general. More precisely, even though all FPT algorithms run in polynomial time if the value of the parameter is some fixed constant, the converse does not necessarily hold. To elaborate on this, we introduce the class XP, that consists of all parameterized decision problems Q for which there exists a computable function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm that decides whether $(x, k) \in Q$ in time $|x|^{f(k)}$. We call such running time XP time and an algorithm running in XP time an XP algorithm. Each problem $Q \in \mathsf{XP}$ can also be solved in polynomial time for each fixed value of the parameter. However, the order of the polynomial depends on the parameter value k, whereas for fixed-parameter tractable problems the order of the polynomial is independent of the value k. To see why we consider FPT to be the class of tractable parameterized problems instead of XP, consider the following two running times $2^k \cdot n^2$ and n^k. While solving an instance of size $n = 1000$ with $k = 10$ may still be feasible with an algorithm with $2^k \cdot n^2$ running time, an n^k algorithm would in this case require more steps than there are stars in the observable universe.

2.3.3 Bounded Search Trees

The method of bounded search trees is fundamental to FPT algorithmic results in a variety of ways. The main idea behind bounded search trees comes from the general idea of backtracking. Basically, an algorithm tries to find a feasible solutions by making guesses, such as for example which endpoint of an edge will go to a vertex cover. Whenever the branching algorithm makes such guess, it *branches* into several subproblems and solves all of them one by one independently. In this manner the execution of a branching algorithm can be viewed as a *search tree*, which is traversed by the algorithm up to the point when a solution is discovered in one of the leaves. In order to justify the correctness of a branching algorithm, one needs to argue that in case of a YES-instance some sequence of decisions captured by the algorithm leads to a feasible solution. If the total size of the search tree is bounded by a function of the parameter alone, and every step takes polynomial time, then such a branching algorithm runs in FPT time. This is indeed the case for many natural backtracking algorithms.

When designing such branching algorithms, we often recursively apply some reduction and branching steps on the instance. A branching rule is an algorithm that takes as an input an istance $I = (x, k)$ of a parameterized problem Q and outputs instances $I_1 = (x_1, k_1), \dots, I_\ell = (x_\ell, k_\ell)$ of the same parameterized problem. We call the integer ℓ the branching factor of the branching rule. A branching rule with branching factor 1 is called a reduction rule. We say that the branching rule is sound, if at least one of the subinstances resulting from the has the same output as the original instance. In particular for decision problems it means that there exists at least one subinstance resulting from the rule which is a YES-instance if and only if the original graph was a YES-instance and for reduction rules this means that the application of the rule preserves the property of being a YES-instance.

2.3.4 Kernelization

The study of kernelization has recently been one of the main areas of research in parameterized complexity, yielding many important new contributions to the theory. Kernelization investigates exact preprocessing algorithms with performance guarantees.

A bikernelization for a parameterized decision problem $P \subseteq \Sigma^* \times \mathbb{N}$ into a parameterized decision problem $Q \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs in time polynomial in $|x| + k$ a pair $(x', k') \in \Sigma^* \times \mathbb{N}$ such that

(i) $(x, k) \in P$ if and only if $(x', k') \in Q$ and

(ii) $|x'| + k' \leq g(k)$, where g is an arbitrary computable function.

The reduced instance (x', k') is the bikernel. If $P = Q$, the reduction is called a kernelization and (x', k') a kernel. The function g is called the *size* of the (bi)kernel, and if g is a polynomial then we say that P admits a *polynomial (bi)kernel*.

The following fact links the existence of bikernels to the existence of kernels.

Fact 2.6 ([9]). *Let P, Q be a pair of decidable parameterized problems such that Q is in* NP *and P is* NP*-hard. If there is a bikernelization from P to Q producing a polynomial bikernel, then P has a polynomial kernel.*

It is relatively easy exercise to see that a parameterized decision problem has a kernel if and only if it is in FPT (for more details see for example Chapter 2 in the book by Cygan et al. [53]). However the size of such kernel can be arbitrary big. Therefore, especially if the problem was already shown to be fixed-parameter tractable, we are mainly interested in the existence of *polynomial kernels*, i.e., kernels whose size is polynomial in k.

The by far most prevalent type of parameter used in kernelization is the *solution size*. Indeed, the existence of polynomial kernels and the exact bounds on their sizes have been studied for a plethora of distinct problems under this parameter, and the rate of advancement achieved in this direction over the past 10 years has been staggering. However, also various structural parameters which have been used to obtain polynomial kernels. A modulator of a graph G to a graph class $\mathcal{H}$ is a vertex set $X \subseteq V(G)$ such that $G - X \in \mathcal{H}$. We denote the cardinality of a minimum modulator to $\mathcal{H}$ in G by $\mathrm{mod}^{\mathcal{H}}(G)$. The vertex cover number of a graph G ($\mathrm{vcn}(G)$) is a special case of $\mathrm{mod}^{\mathcal{H}}(G)$, specifically for $\mathcal{H}$ being the set of edgeless graphs. The vertex cover number has been used to obtain polynomial kernels for problems such as Largest Induced Subgraph [87] and Long Cycle along with other path and cycle problems [27]. Similarly, a feedback vertex set is a modulator to the class of acyclic graphs, and the size of a minimum feedback vertex set has been used to kernelize, for instance, Treewidth [28] and Vertex Cover [128]. Important findings were also obtained in the area of meta-kernelization [26, 89, 138], which is the study of general kernelization techniques and frameworks used to establish polynomial kernels for a wide range of distinct problems.

2.3.5 Fixed-parameter intractability

Just as NP-hardness forms a basic tool to show that the problem is unlikely to have a polynomial algorithm, parameterized complexity has also notion of fixed-parameter intractability. Here the equivalent of the class NP is the class W[1], which is the first class of infinite hierarchy of classes of parameterized decision problems. The W-hierarchy consists of a series of classes W[t], $t \in \mathbb{N}$ such that $\mathsf{W}[t] \subseteq \mathsf{W}[t+1]$ for all t and two additional classes W[SAT] and W[P]. As the precise definitions of these classes are rather technical and not necessary for this thesis, we will not define these classes here and we refer reader for example to the book by Downey and Fellows [64] for the exact definition. However, similarly as in traditional complexity, one can use the concept of reductions to characterize problems that are unlikely to have an FPT algorithm. After, we introduce the notions of parameterized reductions, we will give some natural problems that are complete for classes in W-hierarchy.

Let $Q_1, Q_2 \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized decision problems. A *parameterized reduction* (or FPT reduction) from Q_1 to Q_2 is an FPT algorithm that on an input $(x_1, k_1) \in \Sigma^* \times \mathbb{N}$ produces an output (x_2, k_2) such that

(i) $(x_1, k_1) \in Q_1$ if and only if $(x_2, k_2) \in Q_2$, and

(ii) there is a computable function g such that $k_2 \leq g(k_1)$.

Based on this notion of parameterized reduction, it is straightforward to define the notion of hardness and completeness for these classes in a similar manner as for the class NP. We only need to replace the many-to-one reduction with the parameterized one.

To show that the problem does not have an FPT algorithm unless $\mathsf{FPT} = \mathsf{W}[1]$, we can however use also an equivalent of Turing reduction for traditional complexity called an FPT Turing reduction to some $\mathsf{W}[1]$-hard problem. An FPT Turing reduction from a parameterized problem Q_1 to Q_2 is a deterministic algorithm solving Q_1 with an oracle to Q_2 with the following properties:

(i) the algorithm is FPT, and

(ii) the parameter for Q_2 in each oracle query is bounded by a function of the parameter for Q_1.

As it is strongly believed that $\mathsf{FPT} \neq \mathsf{W}[1]$, to show that the problem is unlikely to have an FPT algorithm, one can simply give a FPT (turing) reduction to some known $\mathsf{W}[1]$-hard problem. Here we list few problems that are often used to show $\mathsf{W}[1]$-hardness and $\mathsf{W}[2]$-hardness, respectively.

Fact 2.7 ([63], Theorem 10.8)**.** CLIQUE *and* INDEPENDENT SET *parameterized by the solution size are both* $\mathsf{W}[1]$*-complete.*

Fact 2.8 ([64], Corollary 23.2.2)**.** DOMINATING SET *and* HITTING SET *parameterized by the solution size are* $\mathsf{W}[2]$*-complete.*

Finally, we introduce one more parameterized class. The class paraNP is defined as the class of decision problems that are solvable by a non-deterministic Turing machine in FPT time. Throughout the thesis, we will make use of the following characterization of paraNP-hardness.

Fact 2.9 ([86], Theorem 2.14)**.** *If a parameterized decision problem Q becomes* NP*-hard after setting the parameter to some fixed constant, then Q is* paraNP*-hard.*

We note that showing paraNP-hardness of a parameterized version of a problem in NP rules out the existence of algorithms with a running time of $\mathcal{O}(n^{f(k)})$. Finally, in Figure 2.1, we depict relations between the parameterized classes mentioned in this section.

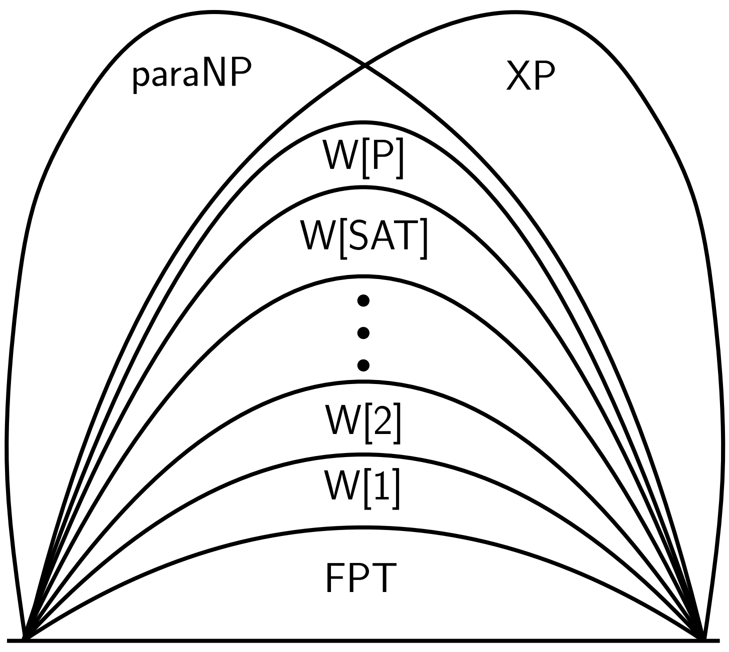


Figure 2.1: The relationships between major complexity classes.

Part I

Decomposition Parameters

CHAPTER 3

Background

This part is devoted to the investigation of the first approach towards the design of structural parameters considered in this thesis, namely, parameters based on decompositions. Decomposition parameters such as *treewidth* [178], *pathwidth* [176], *branchwidth* [179], *clique-width* [49, 50], and *rank-width* [167, 168] play an important role in many modern investigations in algorithmic graph theory and have been a core part of the field of parameterized algorithms already since its early beginnings. The main idea behind decomposition parameters, as the name suggests, is to obtain some kind of decomposition of the input, which in turn helps us solve the given problem more efficiently.

Before we proceed to the main results of this part, we will first introduce several useful decomposition parameter, together with a notion of partially ordered sets that will be crucial in the following chapter. Afterwards, in Chapter 4 we study the possibilities of exploiting treewidth to solve the problem #LinExt. Finally, in Chapter 5 we introduce a novel parameter *prefix pathwidth* that is a restriction of pathwidth which gives rise to an FPT algorithm for QBF.

3.1 Treewidth and Pathwidth

As we already mentioned before, treewidth is a decomposition parameter that measures how "close" the given graph is to a tree, and represents one of the most frequently used tools in parameterized complexity. The notions of treewidth and tree decomposition were introduced by Robertson and Seymour in their fundamental work on graph minors [178]. A tree decomposition of a graph G is a pair $(T, \mathcal{X} = \{X_t\}_{t \in V(T)})$, where T is a rooted tree whose every vertex t is assigned a vertex subset $X_t \subseteq V(G)$, called a *bag*, such that the following properties hold:

(T1) $\bigcup_{t \in V(T)} X_t = V(G)$,

(T2) for every $u \in V(G)$, the set $T_u = \{t \in V(T) : u \in X_t\}$ induces a connected subtree of T (*monotonicity*), and

(T3) for each $uv \in E(G)$ there exists $t \in V(T)$ such that $u, v \in X_t$.

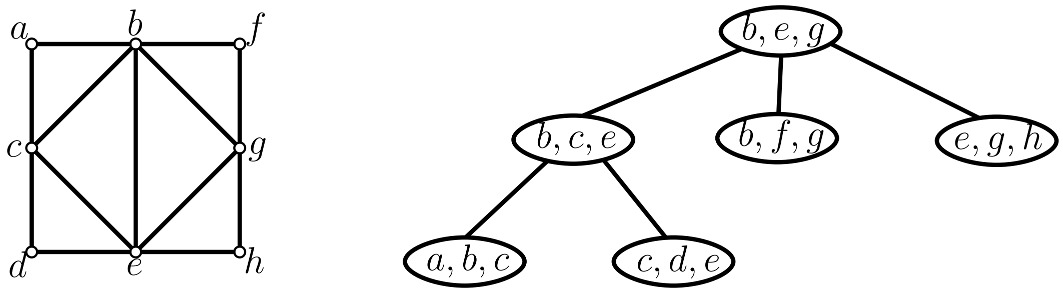


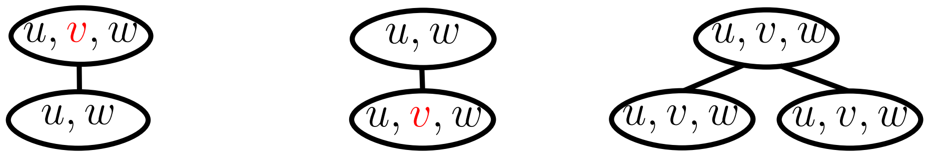
Figure 3.1: A graph (on the left) and its tree decomposition (on the right).

See Figure 3.1 for an example of a tree-composition. To distinguish between the vertices of the tree T and the vertices of the graph G, we will refer to the vertices of T as *nodes*; for brevity, we will also interchange T and $V(T)$ when the context is clear. The *width* of the tree decomposition $\mathcal{T}$ is $\max_{t \in T} |X_t| - 1$. The treewidth of G, $\text{tw}(G)$, is the minimum width over all tree decompositions of G. The purpose of the '-1' in the definition of the width of a decomposition is to let trees have tree-width 1.

In some cases, we will make use of a well-established canonical form of tree decompositions. A tree decomposition $\mathcal{T} = (T, \mathcal{X})$ is *nice* if T contains a root r (introducing natural ancestor-descendant relations in T) and the following conditions are satisfied:

- $X_r = \emptyset$ and $X_\ell = \emptyset$ for every leaf ℓ of T. In other words, all the leaves as well as the root contain empty bags.
- Every non-leaf node of T is of one of the following three types:
  - **Introduce node:** a note t with exactly one child t' such that $X_t = X_{t'} \cup \{v\}$ for some vertex $v \notin X_{t'}$; we say that v is *introduced* at t. If $u \in X_{t'}$ and uv is an edge in G, then we also say that uv is *introduced* at t (Figure 3.2a).
  - **Forget node:** a note t with exactly one child t' such that $X_t = X_{t'} \setminus \{w\}$ for some vertex $w \in X_{t'}$; we say that v is *forgotten* at t (Figure 3.2b).
  - **Join node:** a node t with two children t_1, t_2 such that $X_t = X_{t_1} = X_{t_2}$ (Figure 3.2c).

Nice tree decompositions are in particular useful for designing dynamic programing algorithm along the tree decomposition. To see some simpler examples of such dynamic programing algorithms, the reader is referred for example to the Section 7.3 of the book by Cygan et al. [53]. We note that there exists a polynomial-time algorithm that converts



(a) Introduce (b) Forget (c) Join

Figure 3.2: An example of three types of non-leaf nodes.

an arbitrary tree decomposition into a nice tree decomposition of the same width [141]. A path decomposition is a tree decomposition where each node of T has degree at most 2, and nice path decompositions are nice tree decompositions which do not contain join nodes. Nice path decompositions can also be computed from standard path decompositions in polynomial time while preserving width [141]. Observe that any path decomposition can be fully characterized by the order of appearance of its bags along T, and hence we will consider succinct representations of path decompositions in the form $\mathcal{Q} = (Q_1, \ldots, Q_d)$, where Q_i is the i-th bag in $\mathcal{Q}$. The pathwidth of G, $\mathrm{pw}(G)$, is the minimum width of a path decomposition of G.

We list some useful facts about treewidth and pathwidth.

Fact 3.1 ([22, 30])**.** *There exists an algorithm which, given a graph G and an integer k, runs in time $\mathcal{O}(k^{\mathcal{O}(k^3)}n)$ and either outputs a tree decomposition of G of width at most k or correctly identifies that $\mathrm{tw}(G) > k$. Furthermore, there exists an algorithm which, given a graph G and an integer k, runs in time $\mathcal{O}(k^{\mathcal{O}(k^3)}n)$ and either outputs a path decomposition of G of width at most k or correctly identifies that $\mathrm{pw}(G) > k$.*

Fact 3.2 (Folklore)**.** *Let $\mathcal{T}$ be a tree decomposition of G and $t \in T$. Then each connected component of $G - X_t$ lies in a single subtree of $T - t$. In particular, for each connected component C of $G - X_t$ there exists a subtree T' of $T - t$ such that for each vertex $a \in C$ there exists $t_a \in T'$ such that $a \in X_{t_a}$.*

Fact 3.3 (Courcelle's theorem [46])**.** *Let φ and $\psi = \psi(X)$ be fixed MSO_2 formulas over labeled (directed) graphs. There exists a computable function f and an algorithm such that, given an n-vertex graph G and $S \subseteq V(G)$, decides whether $G \models \varphi$ and whether there exists a set S such that $G \models \psi(S)$ in time $f(\mathrm{tw}(G)) \cdot n$.*

The following extension of the previous fact shows that if G has bounded treewidth then we can count the number of sets S with $G \models \Phi(S)$. This result will be in particular useful in Chapter 4.

Fact 3.4 ([12])**.** *Let $\Phi(X)$ be an MSO_2 formula with a free set variable X and w a constant. Then there is a linear-time algorithm that, given a labeled (directed) graph $G = (V, E)$ of treewidth at most w, outputs the number of sets $S \subseteq E$ such that $G \models \Phi(S)$.*

3.2 Treedepth

The treedepth of a graph is an intuitively easy and very useful invariant. It has been defined by Nešetřil and Mendez [159], but equivalent or similar notions include the *rank function* [161], the *minimum height of an elimination tree* [24], etc. Treedepth has been also studied as a natural parameter for problems that remain W[1]-hard parameterized by treewidth [95, 112].

A treedepth decomposition of a graph G is a pair $(T, \sigma : V(G) \to V(T))$, where T is a rooted forest and σ is a mapping of the vertices of G to the vertices of T such that if $uv \in E(G)$, then either $\sigma(v)$ is an ancestor of $\sigma(u)$ or $\sigma(i)$ is an ancestor of $\sigma(v)$ in T. The *depth* of a treedepth decomposition (T, σ) is the length of a longest path from a leaf to the root of T. Finally the treedepth of a graph G is the depth of a treedepth decomposition with a minimum depth. Note that any depth-first search forest F of G gives a treedepth decomposition, however this decomposition might not be optimal.

Fact 3.5. *Let G be a graph with treedepth d, then it holds that*

$$\mathrm{tw}(G) \leq \mathrm{pw}(G) \leq d - 1 \leq (\mathrm{tw}(G) + 1) \cdot \log(|V(G)|).$$

Fact 3.6 ([160], page 138)**.** *Let $d \in \mathbb{N}$ and $\mathcal{H}$ be the class of graphs of treedepth at most d. Then $\mathcal{H}$ can be characterized by a finite set of forbidden induced subgraphs.*

3.3 Rank-width

Rank-width was introduced by Oum and Seymour [168] and is closely related to clique-width.

Fact 3.7 ([168])**.** *For every graph G it holds that* $\mathrm{rw}(G) \leq \mathrm{cw}(G) \leq 2^{\mathrm{rw}(G)+1}$*, where* $\mathrm{rw}(G)$ *is rank-width of G and* $\mathrm{cw}(G)$ *is clique-width of G.*

To define rank-width, we first need to introduce the bipartite adjacency matrix $\boldsymbol{A}_G[U, W]$. For a graph G and $U, W \subseteq V(G)$, let $\boldsymbol{A}_G[U, W]$ denote the $U \times W$-submatrix of the adjacency matrix over the two-element field GF(2), i.e., the entry $a_{u,w}$, $u \in U$ and $w \in W$, of $\boldsymbol{A}_G[U, W]$ is 1 if and only if $\{u, w\}$ is an edge of G. The *cut-rank* function ρ_G of a graph G is defined as follows: For a bipartition (U, W) of the vertex set $V(G)$, $\rho_G(U) = \rho_G(W)$ equals the rank of $\boldsymbol{A}_G[U, W]$ over GF(2). We note that ρ_G is a symmetric function, and observe that a split-module X can be seen as a subgraph such that $\boldsymbol{A}_G[X, V(G) \setminus X] = 1$.

A rank decomposition of a graph G is a pair (T, μ) where T is a tree of maximum degree 3 and $\mu : V(G) \to \{t : t \text{ is a leaf of } T\}$ is a bijective function. For an edge e of T, the connected components of $T - e$ induce a bipartition (X, Y) of the set of leaves of T. The *width* of an edge e of a rank decomposition (T, μ) is $\rho_G(\mu^{-1}(X))$. The *width* of (T, μ) is the maximum width over all edges of T. The rank-width of G, $\mathrm{rw}(G)$ in short, is the minimum width over all rank decompositions of G. A graph class $\mathcal{H}$ is of *unbounded rank-width* if for each $i \in \mathbb{N}$ there exists a graph $G \in \mathcal{H}$ such that $rw(G) > i$.

An example of a rank decomposition is provided in Figure 3.3.

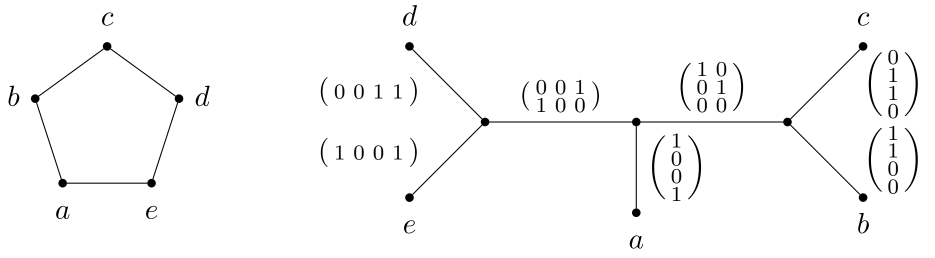

c
b
d
a
e
d
e
a
c
b
$\begin{pmatrix}0&0&1&1\end{pmatrix}$
$\begin{pmatrix}1&0&0&1\end{pmatrix}$
$\begin{pmatrix}0&0&1\\1&0&0\end{pmatrix}$
$\begin{pmatrix}1&0\\0&1\\0&0\end{pmatrix}$
$\begin{pmatrix}1\\0\\0\\1\end{pmatrix}$
$\begin{pmatrix}0\\1\\1\\0\end{pmatrix}$
$\begin{pmatrix}1\\1\\0\\0\end{pmatrix}$

Figure 3.3: A rank decomposition of the cycle C_5.

Theorem 3.8 ([118]). *Let $k \in \mathbb{N}$ and $n \geq 2$. For an n-vertex graph G, we can output a rank decomposition of width at most k or confirm that the rank-width of G is larger than k in time $f(k) \cdot n^3$, where f is a computable function.*

More properties of rank-width can be found, for instance, in [168].

It is known that MSO_1 formulas can be checked in uniformly polynomial time on graphs of bounded rank-width.

Fact 3.9 ([93],[50]). *Let φ and $\psi = \psi(X)$ be fixed MSO_1 formulas. There exists a computable function f and an algorithm such that, given an n-vertex graph G and $S \subseteq V(G)$, decides whether $G \models \varphi$ and whether there exists a set S such that $G \models \psi(S)$ in time $f(\mathrm{rw}(G)) \cdot n^3$.*

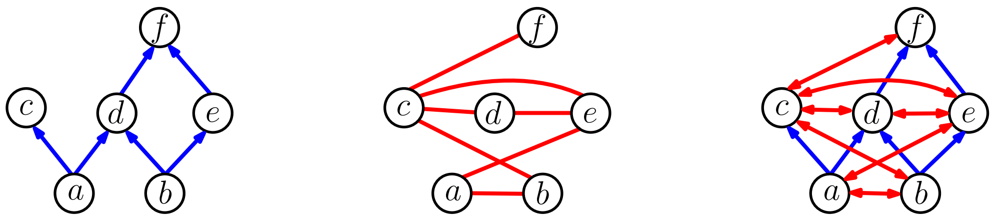
3.4 Posets

A partially ordered set (*poset*) $\mathcal{P}$ is a pair $(P, \leq^P)$ where P is a set and $\leq^P$ is a reflexive, antisymmetric, and transitive binary relation over P. The *size* of a poset $\mathcal{P} = (P, \leq^P)$ is $|\mathcal{P}| := |P|$. We say that p *covers* p' for $p, p' \in P$, denoted by $p' \lhd^P p$, if $p' \leq^P p$, $p \neq p'$, and for every p'' with $p' \leq^P p'' \leq^P p$ it holds that $p'' \in \{p, p'\}$. We say that p and p' are *incomparable* (in $\mathcal{P}$), denoted $p \parallel^P p'$, if neither $p \leq^P p'$ nor $p' \leq^P p$.

A *chain* C of $\mathcal{P}$ is a subset of P such that $x \leq^P y$ or $y \leq^P x$ for every $x, y \in C$. A *chain partition* of $\mathcal{P}$ is a tuple $(C_1, \ldots, C_k)$ such that $\{C_1, \ldots, C_k\}$ is a partition of P and for every i with $1 \leq i \leq k$ the poset induced by W_i is a chain of $\mathcal{P}$. An *antichain* A of $\mathcal{P}$ is a subset of P such that for all $x, y \in A$ it is true that $x \parallel^P y$. A family $C_1, \ldots, C_\ell$ of pairwise disjoint subsets of P forms a total order if for each $i, j \in [\ell]$ and each $a \in C_i$, $b \in C_j$, it holds that $a \leq b$ iff $i < j$. Furthermore, for each $i \in [1, \ell - 1]$ we say that C_i and C_{i+1} are consecutive. The *width* (or *poset-width*) of a poset $\mathcal{P}$, denoted by $\mathrm{width}(\mathcal{P})$ is the maximum cardinality of any anti-chain of $\mathcal{P}$. A subset A of P is downward-closed if for every $a \in A$ it holds that $b \leq^P a \implies b \in A$. For brevity we will often write $\leq^P$ for the poset $\mathcal{P} := (P, \leq^P)$. We call a poset $\mathcal{P}$ such that every two elements of $\mathcal{P}$ are comparable a linear order. A linear extension of a poset $\mathcal{P} = (P, \leq^P)$ is a reflexive, antisymmetric, and transitive binary relation $\preceq$ over P such that $x \preceq y$ whenever $x \leq^P y$ and a poset $\mathcal{P}^* = (P, \preceq)$ is a linear order.

Proposition 3.10 ([81])**.** *Let $\mathcal{P}$ be a poset. Then in time $O(\text{width}(\mathcal{P})\cdot\|\mathcal{P}\|^2)$, it is possible to compute both $\text{width}(\mathcal{P}) = w$ and a corresponding chain partition $(C_1, \dots, C_w)$ of $\mathcal{P}$.*

We consider the following graph representations of a poset $\mathcal{P} = (P, \leq^{\mathcal{P}})$. The cover graph of $\mathcal{P}$, denoted $C(\mathcal{P})$, is the directed graph with vertex set P and edge set $\{(a,b) \mid a \lhd b\}$ (Figure 3.4a). The incomparability graph of $\mathcal{P}$, denoted $I(\mathcal{P})$, is the undirected graph with vertex set P and edge set $\{\{a,b\} \mid a \parallel b\}$ (Figure 3.4b). The combined graph of $\mathcal{P}$, denoted $I_C(\mathcal{P})$, is the directed graph with vertex set P and edge set $\{(a,b) \mid (a \lhd b) \vee (a \parallel b)\}$ (Figure 3.4c); observe that $I_C(\mathcal{P})$ can be obtained by taking disjoint union of the edge sets of $C(\mathcal{P})$ and $I(\mathcal{P})$ and then replacing undirected edges by two directed ones. Finally, the poset graph of $\mathcal{P}$, denoted $P_G(\mathcal{P})$, is the directed graph with vertex set P and edge set $\{(a,b) \mid a \leq b\}$.



(a) Cover graph (b) Incomparability graph (c) Combined graph

Figure 3.4: Graph representations of the poset with elements $\{a,b,c,d,e,f\}$ and the transitive closure of the following relations $a \lhd c, a \lhd d \lhd f, b \lhd e \lhd f$.

We will use the following known fact about tree decompositions and path decompositions of incomparability graphs.

Fact 3.11 ([114, Theorem 2.1])**.** *Let $\mathcal{P}$ be a poset. Then $\text{tw}(I(\mathcal{P})) = \text{pw}(I(\mathcal{P}))$.*

Corollary 3.12 (of Fact 3.1 and 3.11)**.** *Let $\mathcal{P}$ be a poset and $k = \text{tw}(I(\mathcal{P}))$. Then it is possible to compute a nice path decomposition $\mathcal{Q}$ of $I(\mathcal{P})$ of width at most k in time $\mathcal{O}(k^{\mathcal{O}(k^3)}n)$.*

CHAPTER 4

Counting Linear Extensions

Determining the exact number of linear extensions of a given poset is known to be #P-complete [34] already for posets of height at least 3. The currently fastest known method for counting linear extensions of a general n-element poset is by dynamic programming over the lattice of downsets and runs in time $\mathcal{O}(2^n \cdot n)$ [152]. Polynomial time algorithms have been found for various special cases such as series-parallel posets [156] and posets whose cover graph is a (poly)tree [14]. Fully polynomial time randomized approximation schemes are known for estimating the number of linear extensions [68, 35].

Due to the inherent difficulty of the problem, it is natural to study whether it can be solved efficiently by exploiting the structure of the input poset. The first steps in this general direction have been taken, e.g., in [113], using the decomposition diameter as a parameter, in [80] using a parameter called activity for N-free posets, and very recently in [131], where the treewidth of the so-called cover graph was considered as a parameter. Also the exact dynamic programming algorithm [152] can be shown to run in time $\mathcal{O}(n^w \cdot w)$ for a poset with n elements and width w (the size of the largest anti-chain). Interestingly, none of these efforts has so far led to an FPT algorithm.

We believe that this uncertainty about the exact complexity status of counting linear extensions with respect to these various parameterizations is at least partly due to the fact that we deal with a counting problem whose decision version is trivial, i.e., every poset has at least one linear extension. This fact makes it considerably harder to show that the problem is fixed-parameter intractable; in particular, the usual approach based on parsimonious reductions fails. On the other hand, the same predicament makes studying the complexity of counting linear extensions significantly more interesting, as noted also by Flum and Grohe [85]:

The theory gets interesting with those counting problems that are harder than their corresponding decision versions.

Results

Before introducing our results, we provide a formal definition of the problem of counting the number of linear extensions below. We denote the number of linear extensions of $\mathcal{P}$ by $e(\mathcal{P})$.

#LinExt
Instance: A poset $\mathcal{P}$.
Task: Compute $e(\mathcal{P})$.

In this chapter we study the complexity of counting linear extensions when the parameter is the treewidth. In particular, we settle the fixed-parameter (in)tractability of the problem when parameterizing by the treewidth of two of the most prominent graphical representations of posets, the cover graph (also called the Hasse diagram) and the incomparability graph.

Our main result then provides the first evidence that the problem does not allow for an FPT algorithm parameterized by the treewidth of the cover graph unless $\mathsf{FPT} = \mathsf{W}[1]$. We remark that this complements the XP algorithm of Kangas et al. [131] and resolves an open problem recently posed in the Dagstuhl seminar on Exact Algorithms [125]. The result is based on an FPT Turing reduction from Equitable Coloring parameterized by treewidth, and combines a counting argument with a fine-tuned construction to link the number of linear extensions with the existence of an equitable coloring.

We complement this negative result by obtaining an FPT algorithm for the problem when the parameter is the treewidth of the incomparability graph of the poset. To this end, we use the so-called *combined graph* (also called the cover-incomparability graph [33]) of the poset. We employ a special normalization procedure on a decomposition of the incomparability graph to show that the treewidth of the combined graph must be bounded by the treewidth of the incomparability graph. Once this is established, the result follows by giving a formulation of the problem in Monadic Second Order Logic and applying an extension of Courcelle's Theorem for counting.

Organization of the Chapter

The chapter is organized as follows. Section 4.1 is then dedicated to proving the fixed-parameter intractability of the problem when parameterized by the treewidth of the cover graph, and the subsequent Section 4.2 presents our positive results for the problem.

4.1 Fixed-Parameter Intractability of Counting Linear Extensions

The goal of this section is to prove Theorem 4.1, stated below.

Theorem 4.1. #LinExt *parameterized by the treewidth of the cover graph of the input poset does not admit an FPT algorithm unless* W[1]=FPT*.*

The result is based on an FPT Turing reduction from Equitable Coloring parameterized by treewidth of input graph plus number of colors, which is fairly well-known W[1]-hard decision problem.

Fact 4.2 ([78])**.** Equitable Coloring *is* W[1]*-hard parameterized by* $\text{tw}(G) + r$*, where* G *is the input graph and* r *is the number of colors.*

We denote by $\#\text{EC}(G, r)$ the number of equitable colorings of graph G with r colors. The following fact about prime numbers will be useful later in this chapter.

Fact 4.3 ([34])**.** *For any* $n \geq 4$*, the product of primes strictly between* n *and* n^2 *is at least* $n!2^n$*.*

Now, we are ready to give a brief overview of the proof, whose general outline follows the #P-hardness proof of the problem [34]. However, since our parameter is treewidth, we needed to reduce from a problem that is not fixed-parameter tractable parameterized by treewidth. Consequently, instead of reducing from SAT, we will use Equitable Coloring. This made the reduction considerably more complicated and required the introduction of novel gadgets, which allow us to encode the problem without increasing the treewidth too much.

The proof is based on solving an instance (G, r) of Equitable Coloring in FPT time parameterized by $\text{tw}(G) + r$ using an oracle that solves #LinExt in FPT time parameterized by the treewidth of the cover graph (i.e., an FPT Turing reduction). The first step is the construction of an auxiliary poset $\mathcal{P}(G, r)$ of size $2(r-1)|V(G)| + (r^2 - 1)|E(G)|$. Then, for a given sufficiently large (polynomially larger than $|V(G)|$) prime number p, we show how to construct a poset $\mathcal{P}(G, r, p)$ such that $e(\mathcal{P}(G, r, p)) \equiv e(\mathcal{P}(G, r)) \cdot \#\text{EC}(G, r) \cdot A_p \mod p$, where A_p is a constant that depends on p and is not divisible by p. Therefore, if we choose a prime p that does not divide $e(\mathcal{P}(G, r)) \cdot \#\text{EC}(G, r)$, then $e(\mathcal{P}(G, r, p))$ will not be divisible by p. Using Fact 4.3 we show that if $\#\text{EC}(G, r) \neq 0$, then there always exists a prime p within a specified polynomial range of $|V|$ such that p does not divide $e(\mathcal{P}(G, r)) \cdot \#\text{EC}(G, r)$.

From the above, it follows that there exists an equitable coloring of G with r colors if and only if, for at least one prime p within a specified (polynomial) number range, the number of linear extensions of $\mathcal{P}(G, r, p)$ is not divisible by p. Moreover, we show that all inputs for the oracle will have size polynomial in the size of G and treewidth bounded by polynomial in $\text{tw}(G) + r$. Before proceeding to a formal proof of Theorem 4.1, we state two auxiliary lemmas which will be useful for counting linear extensions later in the proof.

Lemma 4.4. *If a poset $\mathcal{P}$ is a disjoint union of posets $\mathcal{P}_1, \ldots, \mathcal{P}_k$ for some positive integer k, then*

$$e(\mathcal{P}) = \frac{(\sum_{i=1}^k |\mathcal{P}_i|)!}{\prod_{i=1}^k |\mathcal{P}_i|!} \prod_{i=1}^k e(\mathcal{P}_i).$$

Proof. We use induction on k, and observe that the lemma trivially holds for $k = 1$. Let $\mathcal{Q}$ denote the disjoint union of posets $\mathcal{P}_1, \ldots, \mathcal{P}_{k-1}$. For each combination of linear extensions of $\mathcal{Q}$ and of $\mathcal{P}_k$ there are $\binom{|\mathcal{Q}|+|\mathcal{P}_k|}{|\mathcal{P}_k|}$ linear extensions of $\mathcal{P}$. Hence,

$$e(\mathcal{P}) = e(\mathcal{Q})e(\mathcal{P}_k)\binom{|\mathcal{Q}|+|\mathcal{P}_k|}{|\mathcal{P}_k|} = \frac{(\sum_{i=1}^{k-1} |\mathcal{P}_i|)!}{\prod_{i=1}^{k-1} |\mathcal{P}_i|!}\left(\prod_{i=1}^{k-1} e(\mathcal{P}_i)\right) \cdot e(\mathcal{P}_k) \cdot \binom{\sum_{i=1}^k |\mathcal{P}_i|}{|\mathcal{P}_k|} =$$

$$\frac{(\sum_{i=1}^{k-1} |\mathcal{P}_i|)!}{\prod_{i=1}^{k-1} |\mathcal{P}_i|!} \frac{(\sum_{i=1}^{k} |\mathcal{P}_i|)!}{(\sum_{i=1}^{k-1} |\mathcal{P}_i|)!|\mathcal{P}_k|!} \prod_{i=1}^k e(\mathcal{P}_i) = \frac{(\sum_{i=1}^k |\mathcal{P}_i|)!}{\prod_{i=1}^k |\mathcal{P}_i|!} \prod_{i=1}^k e(\mathcal{P}_i). \qquad \square$$

Lemma 4.5. *Let p be a prime number and $\mathcal{Q}$ be a connected component of a poset $\mathcal{P}$ such that $|\mathcal{Q}| = p - 1$. If the number of linear extensions of $\mathcal{P}$ is not divisible by p, then the number of elements in each connected component of $\mathcal{P}$ other than $\mathcal{Q}$ is divisible by p.*

Proof. Let $\mathcal{P}_1$ be a connected component of $\mathcal{P}$ different then $\mathcal{Q}$. First note that from Lemma 4.4, it is clear that $e(\mathcal{P})$ will be divisible by the number of linear extensions of the poset $\mathcal{P}'$ formed as a disjoint union of $\mathcal{Q}$ and $\mathcal{P}_1$. Now, by Lemma 4.4 it holds that

$$e(\mathcal{P}') = \frac{(p-1+|\mathcal{P}_1|)!}{(p-1)!|\mathcal{P}_1|!} e(\mathcal{P}_1)e(\mathcal{Q}).$$

Since $e(\mathcal{P})$ is not divisible by p, it must follow that $e(\mathcal{P}')$ is also not divisible by p. Furthermore, $(p-1)!$ cannot be divisible by p since p is prime. Hence it follows that $\frac{(p-1+|\mathcal{P}_1|)!}{|\mathcal{P}_1|!}$ cannot be divisible by p. Suppose that $|\mathcal{P}_1| = ap + b$ for some non-negative integers a and b such that $b < p$; then we obtain that the expression $\frac{(p-1+ap+b)!}{(ap+b)!}$ is not divisible by p. But

$$\frac{(p-1+ap+b)!}{(ap+b)!} = \prod_{i=1}^{p-1}(ap+b+i),$$

which is clearly divisible by $(a+1)p$ whenever $b \geq 1$. Therefore $b = 0$ and hence $|\mathcal{P}_1|$ is divisible by p. $\square$

We now proceed to the proof of our main theorem.

Proof of Theorem 4.1. The proof is structured as follows. We begin by giving the construction of $\mathcal{P}(G, r)$ and $\mathcal{P}(G, r, p)$, after which we establish the desired properties of $\mathcal{P}(G, r, p)$ and $\mathcal{P}(G, r)$, and summarize in the conclusion.

Construction of $\mathcal{P}(G,r)$ and the main gadget. Let (G,r) be an instance of Equitable Coloring such that $|V(G)|$ is divisible by r (if this is not the case, then this can be enforced by padding the instance with isolated vertices, see also [78]). We begin by constructing the poset $\mathcal{P}(G,r)$, which will play an important role later on. For every vertex v of $V(G)$ we create $2(r-1)$ elements denoted $v_{i,j}$, where $1 \le i \le r-1$ and $j \in \{0,1\}$, such that the only dependencies in the poset between these elements are $v_{i,1} \le v_{i,0}$ for all $v \in V(G)$, for all $i \in \{1,\dots,r-1\}$. For every edge $e = uv \in E(G)$ we create r^2-1 pairwise-incomparable elements $e_{i,j}$, such that $(i,j) \in (\{0,\dots,r-1\}^2 \setminus \{(0,0)\})$. The dependencies of $e_{i,j}$ are: if $i > 0$ then $u_{i,0} \le e_{i,j}$, and if $j > 0$ then $v_{j,0} \le e_{i,j}$ (see also Fig. 4.1).

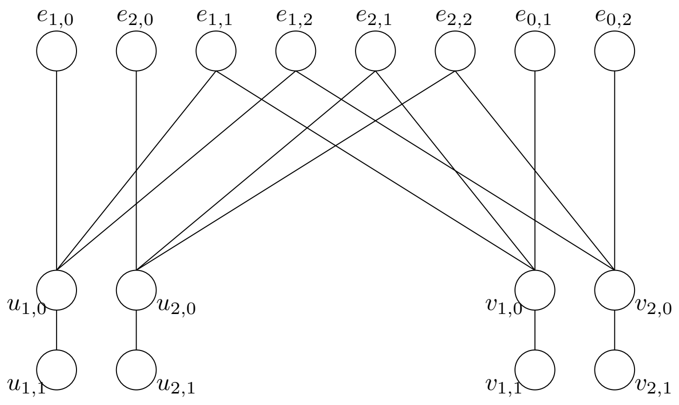


Figure 4.1: The cover graph for an edge $e = uv$ of G in $\mathcal{P}(G,3)$.

Construction of $\mathcal{P}(G,r,p)$. Let us now fix a prime number p such that p does not divide $e(\mathcal{P}(G,r))$ and $p > 2r|V(G)| + r^2|E(G)|$. The main gadget in our reduction is a so-called (a,b)-flower, which consists of an antichain of a vertices (called the petals) covering a chain of $p-b$ elements (called the stalk); an illustration is provided in Fig. 4.2. Due to Lemma 4.5, (a,b)-flowers will later allow us to force a choice of exactly b vertices out of a.

Let G be a graph, r be an integer and p be a prime number as above. Recall that $|V(G)|$ is divisible by r and let $s = \frac{|V(G)|}{r}$ (note that this implies that each color in an equitable coloring of G must occur precisely s times in G). We proceed with a description of the poset $\mathcal{P}(G,r,p)$. The poset $\mathcal{P}(G,r,p)$ is split into $r+3$ "levels" $L_1,\dots,L_{r+3}$ by linearly ordered elements $a_0 \le a_1 \le \dots \le a_{r+2} \le a_{r+3}$, called the anchors. Each of these levels, besides L_{r+3}, will consist of some flowers and a chain of $p-1$ elements which we call a stick; each of these flowers and the stick will always be pairwise incomparable. The anchors a_0 and a_{r+3} are the unique minimum and maximum elements, respectively. The stick and all the stalks of flowers in level L_i will always lie between two consecutive elements a_{i-1} and a_i, and the petals of these flowers will be incomparable with a_i as well as some anchors above that (as defined later). Observe that while the relative position of any stalk and any anchor is fixed in every linear extension, petals can be placed above a_i.

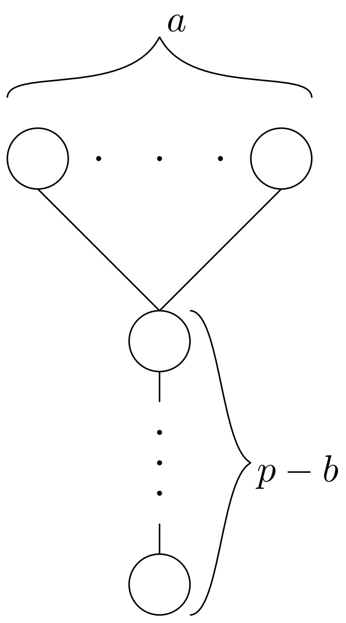


Figure 4.2: An (a, b)-flower.

We say that a flower (or its stalk, petals, or elements) is *associated* with the level in which it is constructed, i.e., with the level L_i such that $a_{i-1} \le c \le a_i$ for stalk elements c and $a_{i-1} \le d$ and $d \parallel a_i$ for petals d. We denote the set of all petals associated with level L_i as A_i (see Fig. 4.3). For the construction, it will be useful to keep in mind the following intended goal: whenever an (a, b)-flower is placed in level i, it will force the selection of precisely b petals (from its total of a petals), where selected elements remain on level i (i.e., between a_{i-1} and a_i) in the linear extension and unselected elements are moved to level $r + 2$ (i.e., between a_{r+2} and a_{r+3}) in the linear extension. We will later show that the total number of linear extensions which violate this goal must be divisible by p, and hence such extensions can all be disregarded modulo p.

The first r levels are so-called *color class* levels, each representing one color class. We use these levels to make sure that every color class contains exactly s vertices. Aside from the stick, each such level contains a single $(|V(G)|, s)$-flower. Recall that the stalk and the stick on level $1 \le i \le r$ both lie between anchors a_{i-1} and a_i, and that the stalk and the flower are incomparable. We associate each petal of the flower at level L_i with a unique vertex $v \in V(G)$ and denote the petal v_i. Each petal v_i will be incomparable with all anchors above a_{i-1} up to a_{r+3}, i.e., $v_i \parallel a_j$ for $i \le j \le r + 2$ and $v_i \le a_{r+3}$. Intuitively, the flower in each color class level will later force a choice of s vertices to be assigned the given color.

Level L_{r+1} is called the *vertex* level and consists of one stick and $|V(G)|$-many $(r, 1)$-flowers; the purpose of this level is to ensure that every vertex is assigned exactly one color. Each flower is associated with one vertex $v \in V(G)$ and we denote the petals of the flower associated with vertex v as v^i for $1 \le i \le r$. We set $v_i \le v^i$ for all

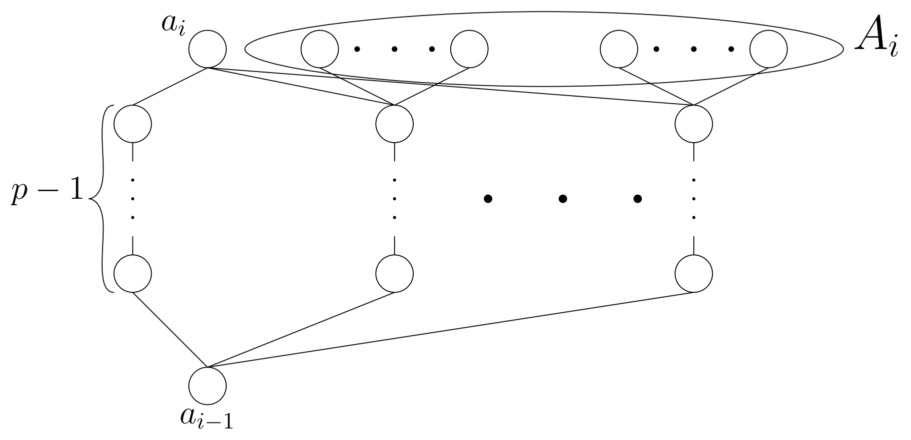


Figure 4.3: Each level consists of a chain of length $p-1$ and a few flowers. The set of petals associated with level L_i is denoted by A_i.

$v \in V(G)$ and $1 \leq i \leq r$.

Level L_{r+2} is called the *edge* level, and its purpose is to ensure that the endpoints of every edge have a different color. It consists of a stick and $|E(G)|$-many $(r^2, 1)$-flowers. Each flower is associated with one edge $e = uv \in V(G)$ and we denote the petals of the flower associated with e as $e_{i,j}$ for $1 \leq i \leq r$ and $1 \leq j \leq r$. Moreover, for edge $e = uv$ we set $u^i \leq e_{i,j}$, $v^j \leq e_{i,j}$, and we set $a_{r+2} \leq e_{i,j}$ whenever $i = j$. Observe that this forces any petal $e_{i,i}$ to lie between a_{r+2} and a_{r+3} in every linear extension (i.e., prevents $e_{i,i}$ from being "selected").

Level L_{r+3} is called the *trash level*. It does not contain any new elements in the poset, but it plays an important role in the reduction: we will later show that any petals which are interpreted as "not selected" must be located between a_{r+2} and a_{r+3} in any linear extension that is not automatically "canceled out" due to counting modulo p.

A high-level overview of the whole constructed poset $\mathcal{P}(G, r, p)$ is presented in Fig. 4.4.

Establishing the desired properties of $\mathcal{P}(G, r, p)$ and $\mathcal{P}(G, r)$. We begin by formalizing the notion of selection. Let a configuration be a partition ϕ of petals of all flowers into $r+3$ sets $L_1^\phi, \dots, L_{r+3}^\phi$. Let Φ denote a set of all configurations. We say that a linear extension $\preceq$ of $\mathcal{P}(G, r, p)$ respects the configuration ϕ if $L_1^\phi \preceq a_1 \preceq L_2^\phi \preceq a_2 \preceq \cdots \preceq a_{r+2} \preceq L_{r+3}^\phi$ and we denote the set of all linear extensions of $\mathcal{P}(G, r, p)$ that respects ϕ by $\mathcal{L}^\phi$. We say that a configuration ϕ is *consistent* if $\mathcal{L}^\phi$ is non-empty; this merely means that $L_1^\phi \leq a_1 \leq L_2^\phi \leq a_2 \leq \cdots \leq a_{r+2} \leq L_{r+3}^\phi$ does not violate any inequalities in $\mathcal{P}(G, r, p)$. Observe that if ϕ is consistent, then $\mathcal{L}^\phi$ is exactly the set of linear extension of the partial order $\mathcal{P}^\phi(G, r, p)$, where $\mathcal{P}^\phi(G, r, p)$ is obtained by enriching $\mathcal{P}(G, r, p)$ with the relations $L_1^\phi \leq a_1 \leq L_2^\phi \leq a_2 \leq \cdots \leq a_{r+2} \leq L_{r+3}^\phi$ and performing transitive closure (in other words, $\mathcal{P}^\phi(G, r, p)$ is obtained by enforcing ϕ onto $\mathcal{P}(G, r, p)$).

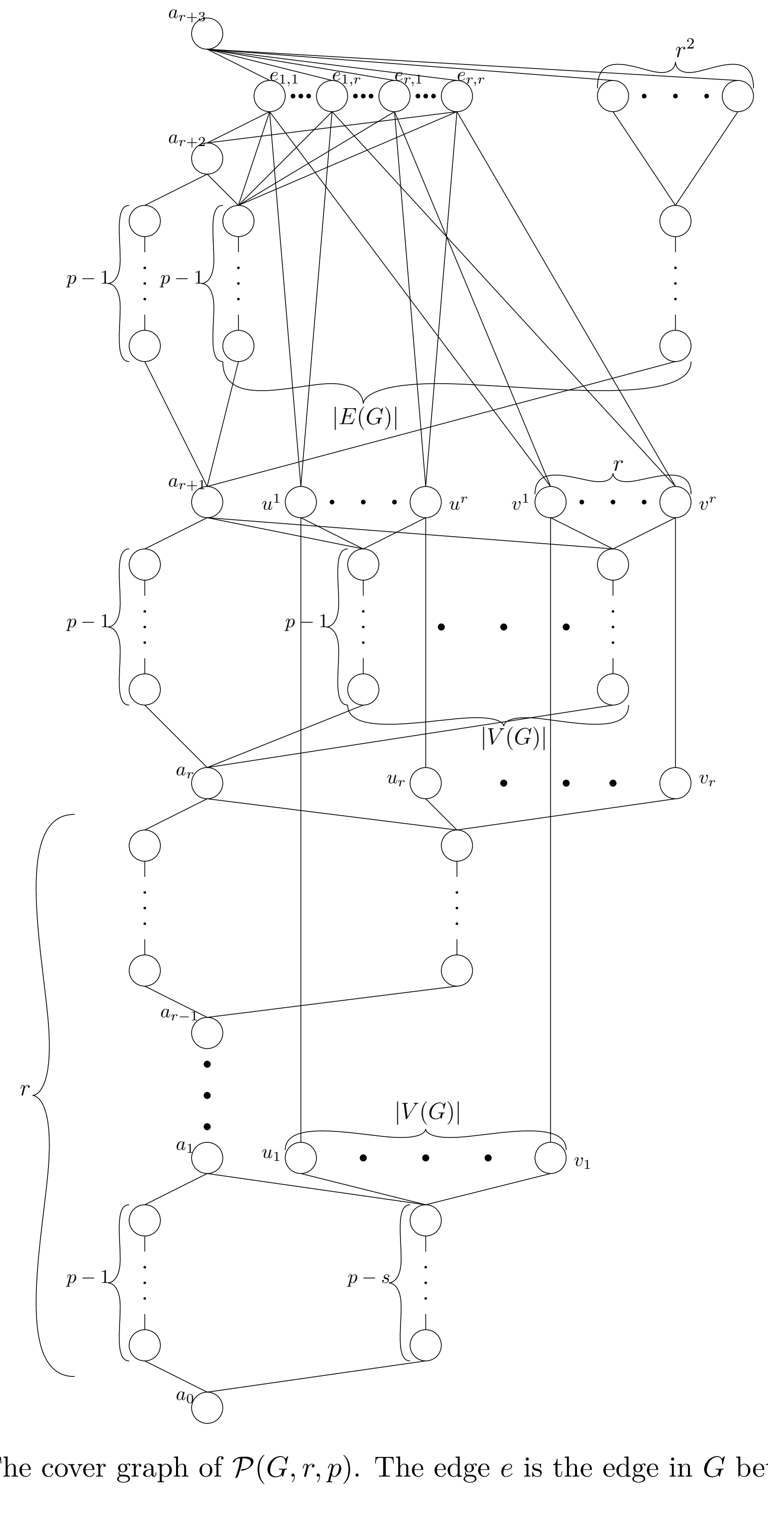


Figure 4.4: The cover graph of $\mathcal{P}(G, r, p)$. The edge e is the edge in G between vertices u and v.

Since every linear extension of $\mathcal{P}(G, r, p)$ respects exactly one configuration, it is easy to see that $e(\mathcal{P}(G, r, p)) = \sum_{\phi \in \Phi} |\mathcal{L}^\phi| = \sum_{\phi \in \Phi} e(\mathcal{P}^\phi(G, r, p))$. Intuitively, a configuration ϕ *contributes* to the above sum modulo p if $e(\mathcal{P}^\phi(G, r, p))$ is not divisible by p. We shall prove that the only configurations which contribute to this sum modulo p are those where from every (a, b)-flower there are exactly b petals in the same level as the stalk, and the remaining $a - b$ petals are in the trash. Furthermore, in each configuration ϕ which contributes to the above sum modulo p, the petals in L^ϕ_{r+1} represent a proper equitable coloring of G with r colors, and each such configuration is respected by the same number of linear extensions.

Let us first remark that for any configuration ϕ, the anchors $a_0, a_1, \ldots, a_{r+3}$ are comparable to all elements of $\mathcal{P}^\phi(G, r, p)$. Now, let $\mathcal{P}^\phi_{L_i}$ be the poset induced by all elements $e \in \mathcal{P}^\phi(G, r, p)$ such that $a_{i-1} \leq e \leq a_i$. It is readily seen that $e(\mathcal{P}^\phi(G, r, p)) = \prod_{i=1}^{r+3} e(\mathcal{P}^\phi_{L_i})$. We proceed by stating a series of claims about our construction.

Claim 4.6. *For each $i \in \{1, \ldots, r\}$, it holds that either $e(\mathcal{P}^\phi_{L_i}) \equiv 0 \mod p$, or $e(\mathcal{P}^\phi_{L_i}) = s!\binom{2p-1}{p}$ and L^ϕ_i contains exactly s petals of A_i and no other petals.*

Proof of the Claim. Assume that $e(\mathcal{P}^\phi_{L_i}) \not\equiv 0 \mod p$ and recall that level L_i contains a stick, which is a chain of $p - 1$ elements that is incomparable with all elements of $\mathcal{P}^\phi_{L_i}$ in every configuration ϕ. By Lemma 4.5 this implies that every connected component of $\mathcal{P}^\phi_{L_i}$ has size divisible by p. Clearly, L^ϕ_i contains only those stalks that are associated with the level L_i, and it contains all such stalks. It is readily seen from the construction that any petal in $\bigcup_{j<i} A_j$ would necessarily form a component of size one in $\mathcal{P}^\phi_{L_i}$. Hence, $\mathcal{P}^\phi_{L_i}$ contains only elements associated with level L_i, namely elements of the chain with $p-1$ vertices and elements of a $(|V(G)|, s)$-flower. Moreover, by Lemma 4.5 and the fact that $|V(G)| + p - s < 2p$, each such flower has exactly p elements in level $\mathcal{P}^\phi_{L_i}$. Since the $p - s$ elements of the stalk must be in $\mathcal{P}^\phi_{L_i}$, the poset $\mathcal{P}^\phi_{L_i}$ contains exactly s elements of A_i. Clearly, the number of linear extensions of the petals of the $(|V(G)|, s)$-flower in $\mathcal{P}^\phi_{L_i}$ is $s!$ and hence by Lemma 4.4 $e(\mathcal{P}^\phi_{L_i}) = s!\binom{2p-1}{p}$.
$\lozenge$

Claim 4.7. *Either $e(\mathcal{P}^\phi_{L_{r+1}}) \equiv 0 \mod p$, or $e(\mathcal{P}^\phi_{L_{r+1}}) = \frac{(|V(G)|p+p-1)!}{(p-1)}!(p!)^{|V(G)|}$ and L^ϕ_{r+1} contains exactly $|V(G)|$ elements of A_{r+1}, specifically one petal for each $(r, 1)$-flower on level L_{r+1}.*

Proof of the Claim. Assume $e(\mathcal{P}^\phi_{L_{r+1}}) \not\equiv 0 \mod p$, and let us first examine elements that are not associated with level L_{r+1}. Clearly, no element associated with level L_{r+2} can appear in $\mathcal{P}^\phi_{L_{r+1}}$ and the only elements associated with any level $i < r + 1$ that can end up in $\mathcal{P}^\phi_{L_{r+1}}$ are petals. Each of these elements is smaller than exactly one petal at level L_{r+1} and independent of all other elements associated with this level. It is easy to see that largest possible size of a connected component of $\mathcal{P}^{\phi}_{L_{r+1}}$ is $p-1+2r<2p$. By Lemma 4.5, every connected component in $\mathcal{P}^{\phi}_{L_{r+1}}$ (except for the stick) will have size p, therefore $\mathcal{P}^{\phi}_{L_{r+1}}$ will contain exactly one element for every antichain associated with L_{r+1} and no other elements. Hence, $\mathcal{P}^{\phi}_{L_{r+1}}$ consists of $|V(G)|$ chains of length p and one chain of length $p-1$. Then $e(\mathcal{P}^{\phi}_{L_{r+1}}) = \frac{(|V(G)|p+p-1)!}{(p-1)}!(p!)^{|V(G)|}$ follows from Lemma 4.4. $\diamondsuit$

Claim 4.8. *Either $e(\mathcal{P}^{\phi}_{L_{r+2}}) \equiv 0 \mod p$, or $e(\mathcal{P}^{\phi}_{L_{r+2}}) = \frac{(|E(G)|p+p-1)!}{(p-1)!(p!)^{|E(G)|}}$ and L^{ϕ}_{r+2} contains exactly $|E(G)|$ elements of A_{r+2}, specifically one petal for each $(r^2,1)$-flower on level L_{r+2}.*

Proof of the Claim. The idea of the proof is similar to the proof of the previous claim, with one additional obstacle: that several flowers can be connected with petals from lower levels into one connected component on level L_{r+2} through the petals of flowers on level L_{r+1}. So, assume $e(\mathcal{P}^{\phi}_{L_{r+2}})$ contains a connected component C which contains at least a single stalk. For each stalk in C, there must be at least one petal in the same flower (otherwise the stalk cannot be connected to the rest of C); in other words, the intersection of each flower and C contains at least p vertices. Let a denote the number of flowers which intersect C, b_2 denote $|A_{r+2} \cap C|$, b_1 denote $|A_{r+1} \cap C|$ and b_0 denote $\sum_{i=1}^{r} |A_r \cap C|$. Then it follows that $|C| = p \cdot a + (b_2 - a) + b_1 + b_0 \le p \cdot a + r^2|E| + r|V| + r|V|$, and recall that $r^2|E| + r|V| + r|V| < p$. Furthermore, if $b_1 > 0$ (and at least one petal from A_{r+1} is required unless C contains only a single flower), we have $a \cdot p < |C| < (a+1) \cdot p$. Hence any such C cannot have size divisible by p and by Lemma 4.5 we have $e(\mathcal{P}^{\phi}_{L_{r+2}}) \equiv 0 \mod p$. Otherwise, if no two flowers are connected through a petal of a flower associated with level L_{r+1}, then every connected component of $\mathcal{P}^{\phi}_{L_{r+2}}$ of size p must consist of a stalk and exactly one petal and the claim follows analogously as the proof of Claim 4.7. $\diamondsuit$

Claim 4.9. *If ϕ is a consistent configuration and for all $i \in \{1, \ldots, r+2\}$ it holds that $e(\mathcal{P}^{\phi}_{L_i}) \not\equiv 0 \mod p$, then the petals in L^{ϕ}_{r+1} encode a proper equitable coloring of $V(G)$ where vertex v receives color i iff the petal v_i lies in L^{ϕ}_i and $\mathcal{P}^{\phi}_{L_{r+3}}$ is isomorphic with $\mathcal{P}(G,r)$.*

Proof of the Claim. From Claims 4.6, 4.7 and 4.8 together with the assumption that $e(\mathcal{P}^{\phi}_{L_i}) \not\equiv 0 \mod p$, it follows that each of the levels $L^{\phi}_1, \ldots, L^{\phi}_r$ contains exactly s petals associated with the corresponding level, level L^{ϕ}_{r+1} contains exactly one petal for each vertex of G and level L^{ϕ}_{r+2} contains exactly one petal for each edge of G.

For the first part of this claim, we observe that each pair of petals in $L_1^\phi, \dots, L_r^\phi$ are associated with distinct vertices of G. If this were not the case, then since $|V(G)| = rs$ there would exist a vertex v such that no element of $L_1^\phi, \dots, L_r^\phi$ is associated with v. But due to the construction at level $r+1$ there exists some $i \in 1, \dots, r$ such that $v^i \in L_{r+1}^\phi$. Then, since $v_i \le v^i$ and v_i can only occur either in level L_i^ϕ or L_{r+3}^ϕ (the latter of which lies above v^i in the linear extension due to the configuration ϕ), this would lead to a contradiction. In particular, we conclude that there is a matching between the petals in level $r+1$ (encoding the color for each vertex) and the union of petals in levels $1, 2, \dots r$ (encoding the vertices assigned to each color class), and by Claim 4.6 it follows that there are exactly s petals in L_{r+1} associated with each color class.

We now argue that the coloring is proper. Observe that by the same argument as above, if an edge $e = uv$ satisfies $e_{i,j} \in L_{r+2}^\phi$, then $u_i \in L_{r+1}^\phi$ and $v_j \in L_{r+1}^\phi$. From the construction of $\mathcal{P}(G, r, p)$ it follows that if $i = j$, then $e_{i,j} \notin L_{r+2}$. Combining these two facts we get that the coloring encoded in L_{r+1}^ϕ is indeed proper.

Now let us take a look at level L_{r+3}^ϕ. To prove the claim, we will construct an isomorphism f from elements of $\mathcal{P}_{L_{r+3}}^\phi$ to elements of $\mathcal{P}(G, r)$. For every vertex $v \in V(G)$, precisely one element $v^i \in L_{r+1}^\phi$ and precisely one of the first r levels contains an element associated with v; to be precise, $v_i \in L_i^\phi$ and $v_j \in L_{r+3}^\phi$ and hence also $v^j \in L_{r+3}^\phi$ for all $j \ne i$. We set $f(v^j) = v_{j,0}$ and $f(v_j) = v_{j,1}$, whenever $j \ne i$ and $j < r$. For the last remaining elements, we set $f(v^r) = v_{i,0}$ and $f(v_r) = v_{r,1}$. Next, for every edge $e = uv$ there is exactly one $e_{a,b} \in L_{r+2}^\phi$. Moreover, if $e_{a,b} \in L_{r+2}^\phi$ then $u_a \in L_{r+1}^\phi$ and $v_b \in L_{r+1}^\phi$, and all other petals for this edge e are in L_{r+3}^ϕ. Let $g_i(r) = i$, $g_i(i) = 0$, and $g_i(k) = k$ otherwise. Then we set $f(e_{i,j}) = e_{g_a(i), g_b(j)}$. Observe that, since $e_{a,b}$ does not lie in L_{r+3}^ϕ, no edge is mapped to the non-existent element $e_{0,0}$ in $\mathcal{P}(G, r)$. It is straightforward to verify that f is really a bijective mapping between elements of $\mathcal{P}_{L_{r+3}}^\phi$ and $\mathcal{P}(G, r)$. Moreover, $f(u) \le f(v)$ in $\mathcal{P}(G, r)$ if and only if $u \le v$ in $\mathcal{P}_{L_{r+3}}^\phi$. Therefore, $\mathcal{P}_{L_{r+3}}^\phi$ is isomorphic with $\mathcal{P}(G, r)$. $\Diamond$

Claim 4.10. *$e(\mathcal{P}(G, r, p)) \not\equiv 0 \mod p$ if and only if $e(\mathcal{P}(G, r)) \cdot \#\mathrm{EC}(G, r) \not\equiv 0 \mod p$.*

Proof of the Claim. From previous claims and in particular Claim 4.9, we already know that $e(\mathcal{P}^\phi(G, r, p)) \ne 0 \mod p$ only if ϕ corresponds to an equitable coloring of G with r colors. Moreover, we know that $e(\mathcal{P}^\phi(G, r, p)) = \prod_{i=1}^{r+3} e(\mathcal{P}_{L_i}^\phi)$ and if $e(\mathcal{P}^\phi(G, r, p)) \not\equiv 0 \mod p$, then

$$\prod_{i=1}^{r+3} e(\mathcal{P}_{L_i}^\phi) = \left(s!\binom{2p-1}{p}\right)^r \frac{(|V(G)|p + p - 1)!}{(p-1)!(p!)^{|V(G)|}} \frac{(|E(G)|p + p - 1)!}{(p-1)!(p!)^{|E(G)|}} e(\mathcal{P}(G, r)).$$

It is readily seen that

$$\left(s!\binom{2p-1}{p}\right)^r \frac{(|V(G)|p + p - 1)!}{(p-1)!(p!)^{|V(G)|}} \frac{(|E(G)|p + p - 1)!}{(p-1)!(p!)^{|E(G)|}}$$

is not divisible by p. We will denote this latter expression as c_p. Hence, it is easy to see that if $\#\mathrm{EC}(G, r)$ denotes actual number of equitable coloring of G with r colors, then

$$\mathcal{P}(G, r, p) \equiv c_p e(\mathcal{P}(G, r))\#\mathrm{EC}(G, r) \mod p.$$

Since c_p is not divisible by p, it is clear that $\mathcal{P}(G, r, p) \not\equiv 0 \mod p$ if and only if $\mathcal{P}(G, r) \cdot \#\mathrm{EC}(G, r) \not\equiv 0 \mod p$. $\Diamond$

Claim 4.11. *If $\#\mathrm{EC}(G, r) \neq 0$, then there is a prime number p greater than $2r|V(G)| + r^2|E(G)|$ and smaller than $(2r|V(G)|+r^2|E(G)|)^2$ such that p does not divide $e(\mathcal{P}(G, r)) \cdot \#\mathrm{EC}(G, r)$.*

Proof of the Claim. Let us first upper bound $e(\mathcal{P}(G, r))\#\mathrm{EC}(G, r)$. Clearly, $\mathcal{P}(G, r)$ contains $m = 2(r - 1)|V(G)| + (r^2 - 1)|E(G)|$ elements, hence $e(\mathcal{P}(G, r)) \leq m!$. It can easily be verified that the number of possibilities of dividing $|V(G)| = rs$ vertices into r color classes with exactly s colors each is $\frac{(rs)!}{(s!)^r}$. By Fact 4.3, the product of all primes between $2r|V(G)| + r^2|E(G)|$ and $(2r|V(G)| + r^2|E(G)|)^2$ is at least $(2r|V(G)| + r^2|E(G)|)!2^{2r|V(G)|+r^2|E(G)|}$. However, $2(r - 1)|V(G)| + (r^2 - 1)|E(G)| + |V(G)| \leq 2r|V(G)| + r^2|E(G)|$ and hence $e(\mathcal{P}(G, r))\#\mathrm{EC}(G, r) \leq (2(r - 1)|V(G)| + (r^2 - 1)|E(G)|)! + \frac{|V(G)|!}{(s!)^r}$ is clearly less than the product of all primes between $2r|V(G)| + r^2|E(G)|$ and $(2r|V(G)| + r^2|E(G)|)^2$. Note that if a natural number N is divisible by set of primes $p_1, \ldots, p_\ell$ then N is divisible by product of these primes and in particular N is bigger than a product of these primes. Therefore, $e(\mathcal{P}(G, r))\#\mathrm{EC}(G, r)$ cannot be divisible by all primes between $2r|V(G)| + r^2|E(G)|$ and $(2r|V(G)| + r^2|E(G)|)^2$. $\Diamond$

Claim 4.12. $\mathrm{tw}(C(\mathcal{P}(G, r, p))) \leq r \cdot (\mathrm{tw}(G) + 3) + 6$.

Proof of the Claim. To distinguish vertices of G and $C(\mathcal{P}(G, r, p))$ in this proof, we will refer to the vertices of $C(\mathcal{P}(G, r, p))$ as elements. So, let $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of G of width $\mathrm{tw}(G)$. Using $\mathcal{T}$, we show how to construct a tree decomposition $\mathcal{T}' = (T', \{X'_t\}_{t \in V(T')})$ of $C(\mathcal{P}(G, r, p))$ with treewidth at most $r \cdot (\mathrm{tw}(G) + 3) + 6$. The construction can be summarized as follows:

1. All bags of $\mathcal{T}'$ will contain the anchors $a_0, \ldots, a_{r+3}$ as well as the top-most element of each stalk of the $(|V(G)|, s)$-flowers in the first r levels; let δ denote this set of $2r + 4$ elements.
2. For every bag $t \in \mathcal{T}$, the tree decomposition $\mathcal{T}'$ will contain a node t' such that if $v \in X_t$ then $\{v^1, \ldots, v^r\} \in X'_t$.

3. Afterwards, every introduce node $t \in \mathcal{T}$ that introduces a vertex v will be replaced by a long path P_t which gradually introduces and subsequently forgets all remaining elements associated with the flower of v at level $r+1$ (i.e., the stalk) as well as every petal from the first r levels associated with v.
4. For each edge e, we pick an introduce node $t \in \mathcal{T}$ which contains both endpoints of e and extend the path P_t by a new segment which introduces and subsequently forgets all elements associated with the flower of e at level $r+2$.
5. The root node is replaced by a path that takes care of all elements which are not associated with any vertex or edge in G.

Let us now take a closer look what happens in $\mathcal{T}'$ when $t \in T$ is an introduce node. Let $v \in V(G)$ be the vertex introduced at t. Since X'_t contains elements $a_0, \ldots, a_{r+2}, v^1, \ldots, v^r$, and the maximum element of each stalk of the $(|V(G)|, s)$-flower in the first r levels, it is easily seen that every petal from the first r levels associated with v as well as the stalk of the flower associated with v in level L_{r+1} each forms a separate connected component of $C(\mathcal{P}(G,r,p)) \setminus X'_t$. Moreover, for an edge $e = uv$ such that X_t contains both endpoints of e, we have that X'_t also contains elements $u^1, \ldots, u^r$ and one can see that also the flower associated with e at level L_{r+2} is a connected component of $C(\mathcal{P}(G,r,p)) \setminus X'_t$. It is readily seen that the singleton, chain, and flower all have pathwidth 1 and hence there is a nice path decomposition $(B_1, \ldots, B_{\ell_t})$ with $B_1 = B_{\ell_t} = \emptyset$ of the graph containing every petal element from first r levels associated with v, the stalk of the flower associated with v in level L_{r+1}, and the flower associated with e in level L_{r+2} for every edge e introduced at t. We then replace X'_t by a path $(Y_1, \ldots, Y_{\ell_t})$ such that $Y_i = B_i \cup X'_t$ and for each $i \in \{1, \ldots, \ell_t - 1\}$ the node with bag Y_{i+1} is the parent of the node with the bag Y_i.

It is easy to see now, that when we are forgetting vertex v in node t in $\mathcal{T}$, we can forget element $v^1, \ldots, v^r$ in $\mathcal{T}'$, because we already introduced all its adjacent edges in $C(\mathcal{P}(G,r,p))$ either in the path corresponding to the node of $\mathcal{T}$ introducing v or the one introducing a neighbor of v.

Finally, when we get to the root node of $\mathcal{T}$, we have already forgotten all elements associated with any specific vertex or edge of G. Therefore, the only elements besides δ which need to be included in $\mathcal{T}'$ are the remaining elements in the stalks in the first r levels and the sticks in every level. However, it is easy to see that at this point they all form separate chains in $C(\mathcal{P}(G,r,p)) \setminus \delta$. Hence there once again exists a path decomposition of width at most $|\delta| + 2$ which gradually introduces and subsequently forgets all of these elements.

One can readily see that the properties (T1), (T2), and (T3) are satisfied and we are only left with computing the width of $\mathcal{T}'$. By construction, every join and forget node in $\mathcal{T}$ will become a node in $\mathcal{T}'$ whose bag has size at most $r \cdot |X_t| + 2r + 4$. On the other hand, every introduce node in $\mathcal{T}$ will become a path in $\mathcal{T}'$, and the largest bag on this path has size at most $r \cdot |X_t| + 2r + 6$. $\Diamond$

Concluding the proof. Let us summarize the FPT Turing reduction used to prove Theorem 4.1. Given an instance (G, r) of Equitable Coloring, we loop over all primes p such that $2r|V(G)| + r^2|E(G)| < p < (2r|V(G)| + r^2|E(G)|)^2$, and for each such prime we construct the poset $\mathcal{P}(G, r, p)$; from Claim 4.11 it follows that if $\#\mathrm{EC}(G, r) \neq 0$, then at least one such prime will not divide $e(\mathcal{P}(G, r)) \cdot \#\mathrm{EC}(G, r)$, and by Claim 4.12 each of the constructed posets $\mathcal{P}(G, r, p)$ has bounded treewidth of the cover graph. For each such poset $\mathcal{P}(G, r, p)$, we compute $e(\mathcal{P}(G, r, p))$ by the black-box procedure provided as part of the reduction. If for any prime p we get $e(\mathcal{P}(G, r, p)) \not\equiv 0 \mod p$, then we conclude that (G, r) is a yes-instance, and otherwise we reject (G, r), and this is correct by Claim 4.10. □

Remark. Note that the reduction above does not use the full power of the Turing reduction. Namely, the calls to the oracle are independent of each other and could be asked at the same time. Such a Turing reduction is also sometimes called a truth-table reduction.

4.2 Fixed-Parameter Tractability of Counting Linear Extensions

This section is dedicated to proving our algorithmic result, stated below.

Theorem 4.13. *#LinExt is fixed-parameter tractable parameterized by the treewidth of the incomparability graph of the input poset.*

The proof of Theorem 4.13 is divided into two steps. First, we apply a transformation process to a path decomposition $\mathcal{Q}$ of small width (the existence of which is guaranteed by Corollary 3.12) of $I(\mathcal{P})$ which results in a tree decomposition $\mathcal{T}$ of $I(\mathcal{P})$ satisfying certain special properties. The properties of $\mathcal{T}$ are then used to prove that $I_C(\mathcal{P})$ has treewidth bounded by the treewidth of $I(\mathcal{P})$. In the second step, we construct an MSO_2 formulation which enumerates all the linear extensions of $\mathcal{P}$ using $I_C(\mathcal{P})$, and apply Fact 3.4.

4.2.1 The Treewidth of Combined Graphs

We begin by arguing a useful property of separators in incomparability graphs.

Lemma 4.14. *Let $S \subseteq V(I(\mathcal{P}))$. Then for each pair of distinct connected components C_1, C_2 in $I(\mathcal{P}) - S$, it holds that for any $a_1, b_1 \in C_1$ and any $a_2, b_2 \in C_2$ we have $a_1 \leq a_2$ iff $b_1 \leq b_2$. Namely, the poset contains a total order of all connected components in $I(\mathcal{P}) - S$.*

Proof. See Figure 4.5 for an illustration of the proof. We begin by proving the following claim.

Claim 4.15. *Let a, b, c be three distinct elements of $\mathcal{P}$ such that $a \parallel b$ and both pairs a, c and b, c are comparable. Then $a \leq c$ iff $b \leq c$.*

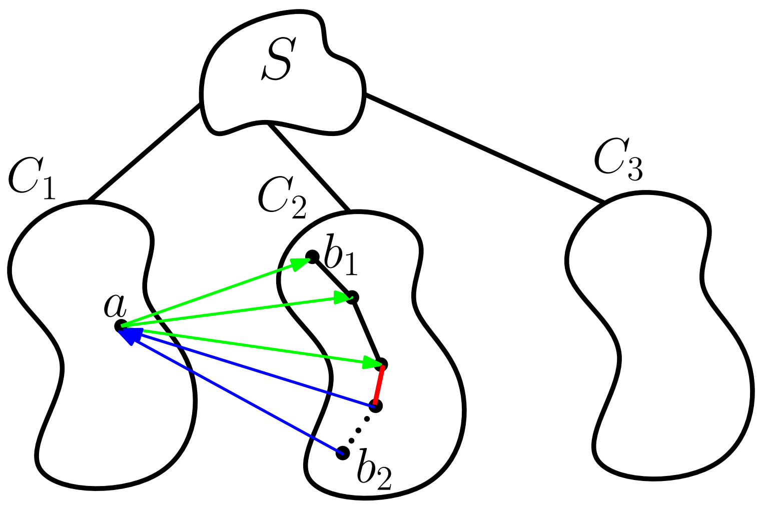


Figure 4.5: Incomparability graph G with a separator S and components C_1, C_2, and C_3 or $G - S$. The elements b_1 and b_2 in C_2 has to be either both smaller or bigger than a, otherwise the red edge would not be in the incomparability graph.

Proof of the Claim. Suppose that, w.l.o.g., $a \leq c$ and $c \leq b$. Then by the transitivity of $\leq$, we get $a \leq b$ which contradicts our assumption that $a \parallel b$. ◊

Now to prove Lemma 4.14, assume for a contradiction that, w.l.o.g., there exist $a_1, b_1 \in C_1$ and $a_2, b_2 \in C_2$ such that $a_1 \leq b_1$ and $b_2 \leq a_2$. Let Q_1 be an a_1-a_2 path in $I[C_1]$. By Claim 4.15, $a_1 \leq b_1$ implies that every element q on Q_1 satisfies $q \leq b_1$, and in particular $a_2 \leq b_1$. Next, let Q_2 be a b_1-b_2 path in $I[C_2]$. Then Claim 4.15 also implies that each element q' on Q_2 satisfies $a_2 \leq q'$. Since b_2 lies on Q_2, this would imply that $a_2 \leq b_2$, a contradiction. □

To proceed further, we will need some additional notation (see Figure 4.6). Let $\mathcal{T} = (T, \mathcal{X})$ be a rooted tree decomposition and $t \in T$. We denote by $L(t)$ the set of all vertices which occur in the "branch" of $T - t$ containing the root r; formally, $L(t) = \{v \in X_{t'} \setminus X_t \mid t'$ lies in the same connected component as r in $T - t\}$. We then set $R(t) = V(G) \setminus (L(t) \cup X_t)$. We also let T_t^r denote the connected component of $T - t$ which contains the root r.

Next, recall that each connected component of the graph obtained after deleting X_t must lie in a subtree of $T - t$ (Fact 3.2). A block of a bag X_t in a rooted tree decomposition $\mathcal{T} = (T, \mathcal{X})$ is a sequence of consecutive connected components in $(I(\mathcal{P}) - X_t) \cap R(t)$ (see Figure 4.7 for an illustration). We say that a node $t \in T$ has z blocks if there exist z distinct blocks of X_t. Blocks will play an important role in the tree decomposition we wish to obtain from our initial path decomposition of $I(\mathcal{P})$. The following lemma captures the operation we will use to alter our path decomposition.

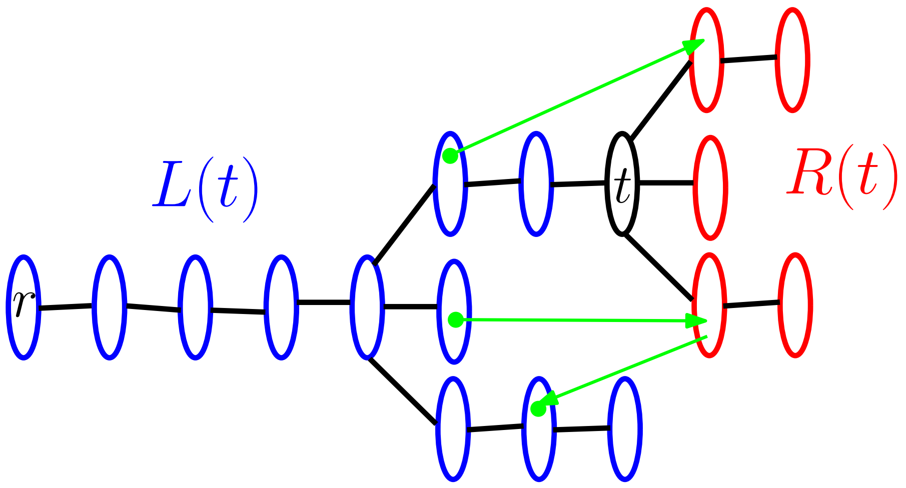


Figure 4.6: A tree decomposition of the incomparability graph with root r. For the bag t the set $L(t)$ are the vertices that appear in some of the blue bags but not in t. Similarly, $R(t)$ is the set of vertices that appear in the red bags but not in t. The green vertices are the vertices in $L(t)$ with an cover edge to $R(t)$. After Lemma 4.17, we call them cover-guard of t. The main point of the whole section is to show that we can obtain a special type of tree decomposition of the incomparability graph, which has only small number of cover-guards for every t that can be added to t to abtain a tree decomposition of the combined graph of a small width.

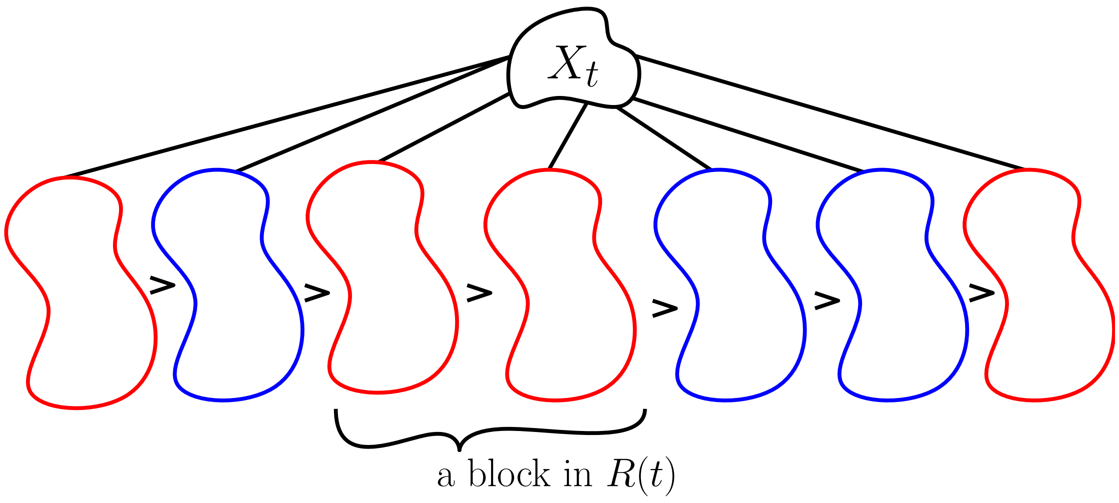


Figure 4.7: A bag X_t in a tree decomposition of the incomparability graph. The blue 'blobs' represent the components of $G - X_t$ with vertices in $L(t)$ and the red ones with the vertices in $R(t)$. The maximal consecutive sets of red components are blocks. Note that there are no cover edges between different blocks.

Lemma 4.16. *Let $\mathcal{T} = (T, \mathcal{X})$ be a rooted tree decomposition of a graph G and let $t \in T$ be such that there are z blocks of X_t. Then there is a tree decomposition $\mathcal{T}'(T', \mathcal{X}')$ satisfying:*

1. *The width of $\mathcal{T}'$ is at most the width of $\mathcal{T}$.*
2. *The tree T' contains T_t^r as a subtree which is separated from the rest of T' by t.*
3. *The degree of t in T' is $z + 1$.*
4. *There exists a bijection α between the z blocks of X_t and the z trees in $T' - t$ other than T_t^r such that for each block B of X_t, we have $\bigcup_{s \in \alpha(B)} X'_s \setminus X_t = B$.*
5. *For each $t' \in N[t] \setminus T_t^r$, we have $X_{t'} = X_t$.*

Proof. It will be useful to observe that $T - T_t^r$ is a subtree of T and in particular it is connected. Consider the following construction of $\mathcal{T}'$. First, we copy all nodes of $T_t^r \cup \{t\}$ (along with their bags) into $\mathcal{T}'$, thus ensuring that Property **2** holds. Second, for each block B of X_t we make a copy T^B of the tree $T - T_t^r$, and connect the node t^B corresponding to t in $T \setminus T_t^r$ by an edge to the node t in T'. Moreover, for each node $s \in T \setminus T_t^r$ we set $X'_{s^B} = X_s \cap (B \cup X_t)$. It is easy to verify that all of the required properties are now satisfied, and it remains to show that $\mathcal{T}'$ is indeed a tree decomposition.

We argue that $\mathcal{T}'$ satisfies all three properties of tree decompositions. Property (T1) follows directly from fact that $\mathcal{T}$ was tree decomposition, and hence every vertex that does not occur in a bag in T_t^r must occur in some bag X_s for some node $s \in T \setminus T_t^r$; then this vertex either also occurs in X_t or occurs in some block B and hence in X'_{s^B}. Property (T2) is also straightforward, since each vertex either does not occur in any block or in exactly one block, and in both cases monotonicity follows from the monotonicity of $\mathcal{T}$ and the construction. For the final Property (T3), we recall that there are no edges between the blocks of X_t; in particular every edge $e = ab$ in $I(\mathcal{P})[X_t \cup R(t)]$ is either contained in X_t, goes between a vertex of X_t and a vertex of some block B, or is contained in some block B. In all three cases, it holds that if $a, b \in X_s$ for some $s \in T \setminus T_t^r$, then $e \in X_{s^B}$ for some block B. Therefore, $\mathcal{T}'$ is a tree decomposition. □

We proceed by showing how we can apply Lemma 4.16 to transform a given path decomposition.

Lemma 4.17. *Let $\mathcal{Q}$ be a nice path decomposition of $I(\mathcal{P})$. Then there is a rooted tree decomposition $\mathcal{T} = (T, \mathcal{X})$ of $I(\mathcal{P})$ with the following properties. $\mathcal{T}$ is rooted at a leaf r and $X_r = \emptyset$, the width of $\mathcal{T}$ is at most the width of $\mathcal{Q}$, and for any node $t \in T$ with $z > 1$ blocks:*

1. *The degree of t in T is $z + 1$.*

2. *There exists a bijection α between the z blocks of X_t and the z trees in $T' - t$ other than T_t^r such that for each block B of X_t, we have $\bigcup_{s\in\alpha(B)} X_s \setminus X_t = B$.*

3. *For $t' \in N(t) \cap T_t^r$ there exists a vertex v such that $X_{t'} = X_t \setminus \{v\}$, and furthermore t' has degree 2 and 1 block.*

4. *For each pair of neighbors $t, t' \in T$, it holds that $|X_t \setminus X_{t'}| + |X_{t'} \setminus X_t| \leq 1$.*

Proof. Let us order the vertices of $I(\mathcal{P})$ in the order in which they were introduced in $\mathcal{Q}$. We set the first leaf of $\mathcal{Q}$ to be a root, and observe that $X_r = \emptyset$ since $\mathcal{Q}$ is nice. We then process the vertices of $I(\mathcal{P})$ in their order of introduction; when processing each such vertex v, we apply Lemma 4.16 to the unique node t of the current tree decomposition which is closest to r and contains v; with a slight abuse of terminology, we say that t is the node where v is introduced. We show that the following invariants hold after (and before) each step of this procedure:

1. For each pair of neighbors $t, t' \in T$, it holds that $|X_t \setminus X_{t'}| + |X_{t'} \setminus X_t| \leq 1$.
2. For each vertex u that was already processed, the introduce node of u satisfies the conditions of the lemma.
3. Any node t of degree greater than 2 is an introduce node of an already processed vertex.

Clearly, all invariant conditions holds for $\mathcal{Q}$ rooted at r. For the induction step, suppose that the conditions hold in a tree decomposition $\mathcal{T}$ obtained by inductively applying Lemma 4.16 and the first unprocessed vertex is v. Let t be the unique node where v is introduced, and let $\mathcal{T}'$ be the tree decomposition we obtained by applying Lemma 4.16 on $\mathcal{T}$ and t.

It is easy to verify that $\mathcal{T}'$ then satisfies the desired Conditions **1**, **2**, and **4** at the node t by Lemma 4.16. As for Condition **3**, since t is the introduce node of v and the first invariant condition holds in $\mathcal{T}$, it is clear that for $t' \in N(t) \cap T_t^r$ it is the case that $X_{t'} = X_t \setminus \{v\}$. Moreover, t' cannot be an introduce node, since then t' would have to introduce an already processed vertex, which would imply that $X_{t'} = X_t$. So, let us consider the node s on the unique t'-r path that is the closest introduce node to t', and let s' be the neighbor of s on the s-t' path. Since no vertex was introduced on the s'-t' path, it follows that $R(s') = R(t')$. Since s' only has 1 block by the construction, it must be the case that t' also only has one block, and so Condition **3** also holds.

We proceed by arguing that the invariant conditions remain satisfied by $\mathcal{T}'$. Since $\mathcal{Q}$ was nice and $\mathcal{T}$ satisfied the first invariant condition, it is readily seen that the first invariant condition holds for all pairs of neighbors in T_t^r as well as for t with all of its neighbors. If s^B and s'^B are neighbors in a tree $\alpha(B)$ of $T' - t$, then by the construction in Lemma 4.16 there exists a pair of neighbors $s, s' \in T \setminus T_t^r$ such that $X'_{s^B} = X_s \cap (B \cup X_t)$ and $X'_{s'^B} = X_{s'} \cap (B \cup X_t)$. But then $|X'_{s^B} \setminus X'_{s'^B}| + |X'_{s'^B} \setminus X'_{s^B}| = |(X_s \cap (B \cup X_t)) \setminus$

$(X_{s'} \cap (B \cup X_t))| + |(X_{s'} \cap (B \cup X_t)) \setminus (X_s \cap (B \cup X_t))| \leq |X_s \setminus X_{s'}| + |X_{s'} \setminus X_s| \leq 1$ and the first invariant condition follows. As for the second invariant condition, notice that from the construction it follows that all the vertices that precede v in the order of introduction in $\mathcal{Q}$ must have been introduced in some node of T_t^r, and the application of Lemma 4.16 does not alter such introduce nodes for previously processed vertices. Finally, by the induction hypothesis all nodes of $T \setminus T_t^r$ have degree at most 2, therefore from the construction in Lemma 4.16 it is clear that all nodes in $T' \setminus (T_t^r\{t\})$ also have degree 2. Since all introduce nodes of unprocessed vertices lie in $T' \setminus (T_t^r\{t\})$, we conclude that the third invariant condition also holds in $\mathcal{T}'$.

Now, let us consider the tree decomposition $\mathcal{T}$ obtained after processing all vertices of $I(\mathcal{P})$ according to the procedure described above. $\mathcal{T}$ satisfies Condition **4** due to the first invariant of our procedure, and for all other conditions it suffices to consider nodes with more than 1 block. In particular, it suffices to verify that all such nodes satisfy the conditions of Lemma 4.16 and additionally also condition **3** of this Lemma. So, suppose for a contradiction that there exists a node t which does not meet these conditions, but all nodes on the unique t-r path do. Then there are two possibilities to consider for the unique neighbor t' of t on the t-r path. If t' were to have more than 1 block, then by our assumptions t' would have to satisfy the conditions of Lemma 4.16, contradicting the fact that t has more than 1 block. On the other hand, if t' were to have only a single block, then by construction t must be an introduce node of some vertex v and by our invariants and the construction it follows that t in fact must satisfy all the required conditions. In particular we conclude that all nodes t satisfy Conditions **1-4**. $\square$

We call a tree decomposition rooted at a leaf with $X_r = \emptyset$ which satisfies the properties of Lemma 4.17 a blocked tree decomposition. The next ingredient we will need for proving that $I_C(\mathcal{P})$ has small treewidth is the notion of *cover-guards* (these are the green vertices in Figure 4.6).

Let $\mathcal{T} = (T, \mathcal{X})$ be a tree decomposition of $I(\mathcal{P})$ rooted at r and let $t \neq r$. Then the cover-guard of t, denoted $\mathcal{A}_t$, is the set of vertices in $L(t)$ which are incident to a cover edge whose other endpoint lies in $R(t)$; formally,

$$\mathcal{A}_t = \{v \in L(t) \mid \exists u \in R(t) : (uv \in E(C(\mathcal{P})) \vee vu \in E(C(\mathcal{P})))\}.$$

For a vertex $v \in I(\mathcal{P})$, we let $\mathcal{A}^v = \{t \in T \mid v \in \mathcal{A}_t\}$ and $X^v = \{t \in T \mid v \in X_t\}$.

Our next aim is to add all the cover-guards into each bag. The following lemma will allow us to argue that the result is still a tree decomposition; it is worth noting that the assumption that the decomposition is blocked is essential for the lemma to hold.

Lemma 4.18. *Let $\mathcal{T} = (T, \mathcal{X})$ be a blocked tree decomposition of $I(\mathcal{P})$ rooted at r and let $v \in I(\mathcal{P})$. Then $T[\mathcal{A}^v \cup X^v]$ is a tree.*

Proof. Since $\mathcal{T}$ is a tree decomposition, we have that $T[X^v]$ must be a tree. Consider a connected component A of $T[\mathcal{A}^v]$ and its unique A-X^v path Q, with endpoints $x \in X^v$

and $a \in A$. Since r is located at a leaf of T it must hold that $r \notin Q$. We consider two cases: either r lies in the same connected component as X^v in $T - Q$, or it lies in a different connected component.

In the former case, it follows that each internal vertex q of Q satisfies $R(q) \subseteq R(a)$ and $v \in L(q)$. But then by the definition of $\mathcal{A}^v$ and the fact that $a \in \mathcal{A}^v$, this would imply $q \in \mathcal{A}^v$, contradicting our construction of Q. Hence if r lies in the same connected component as X^v in $T - Q$, then A is adjacent to X^v.

In the latter case, there must exist a node $q \in Q$ of degree at least 3 such that each of A, X^v and r occur in different components of $T - q$. By the definition of $\mathcal{A}^v$, there exists a vertex $u \in R(a)$ such that $v \lhd^{\mathcal{P}} u$ or $u \lhd^{\mathcal{P}} v$. Since $u, v \in R(q)$ due to the location of the root and there is a cover edge between them, it follows that either u, v occur in the same connected component of X_q or in two consecutive ones, but in either case u, v must lie in the same block of q, say block B. But since $u, v \notin X_q$, this contradicts Property **2** in Lemma 4.17; indeed, each tree in $T - q$ contains at most one of v, u in its bags, and hence there exists no tree T' in $T - q$ satisfying $\bigcup_{t' \in T'} X_{t'} \setminus X_q = B$. Hence r cannot occur in a different connected component than X^v in $T - Q$.

We conclude that Q contains no internal vertices. In particular, every connected component of $T[\mathcal{A}^v]$ is adjacent to X^v. □

Lemma 4.19. *Let $\mathcal{T} = (T, \mathcal{X})$ be a blocked tree decomposition of $I(\mathcal{P})$ of width k. Then for each $t \in T$ it holds that $|\mathcal{A}_t| \leq 2k + 2$.*

Proof. First, observe that if a node $t \in T$ has 0 blocks, then $R(t) = \mathcal{A}_t = \emptyset$. So, consider a node t which has exactly 1 block consisting of connected components $(D_1, \ldots, D_j)$ in $(I(\mathcal{P}) - X_t) \cap R(t)$.

Claim 4.20. $|\mathcal{A}_t| \leq 2k + 2$.

Proof of the Claim. For an illustration of the proof see Figure 4.8. Assume for a contradiction that $|\mathcal{A}_t| > 2k + 2$. By Lemma 4.14 we have that $(D_1, \ldots, D_j)$ are consecutive connected components in a total order of connected components in $I(\mathcal{P}) - X_t$. Hence any edge in $C(\mathcal{P}) - X_t$ between $R(t)$ and $L(t)$ must necessarily have one endpoint in $D_1 \cup D_j$. Furthermore, an element in $\mathcal{A}_t$ cannot be adjacent to both D_1 and D_j in $C(\mathcal{P}) - X_t$ due to transitivity and acyclicity. So, we may partition $\mathcal{A}_t$ into $\mathcal{A}_t^1 = \{v \in \mathcal{A}_t \mid \exists u \in D_1 : v \lhd^{\mathcal{P}} u\}$ and $\mathcal{A}_t^2 = \{v \in \mathcal{A}_t \mid \exists u \in D_j : u \lhd^{\mathcal{P}} v\}$.

By Lemma 4.14, it also follows that $\mathcal{A}_t^1$ and $\mathcal{A}_t^2$ must each lie in separate connected components of $I(\mathcal{P}) - X_t$, say C^1 and C^2, respectively. Furthermore, each element in $\mathcal{A}_t^1$ is maximal in C^1 and each element in $\mathcal{A}_t^2$ is minimal in C^2. In particular, each of $\mathcal{A}_t^1$, $\mathcal{A}_t^2$ forms a clique in $I(\mathcal{P})$. But by our assumption on the size of $\mathcal{A}_t$, at least one of $\mathcal{A}_t^2$ and $\mathcal{A}_t^1$ must have size greater than $k+1$, which implies that $I(\mathcal{P})$ contains a clique of size at least $k + 2$. It is well-known that each clique must be completely contained

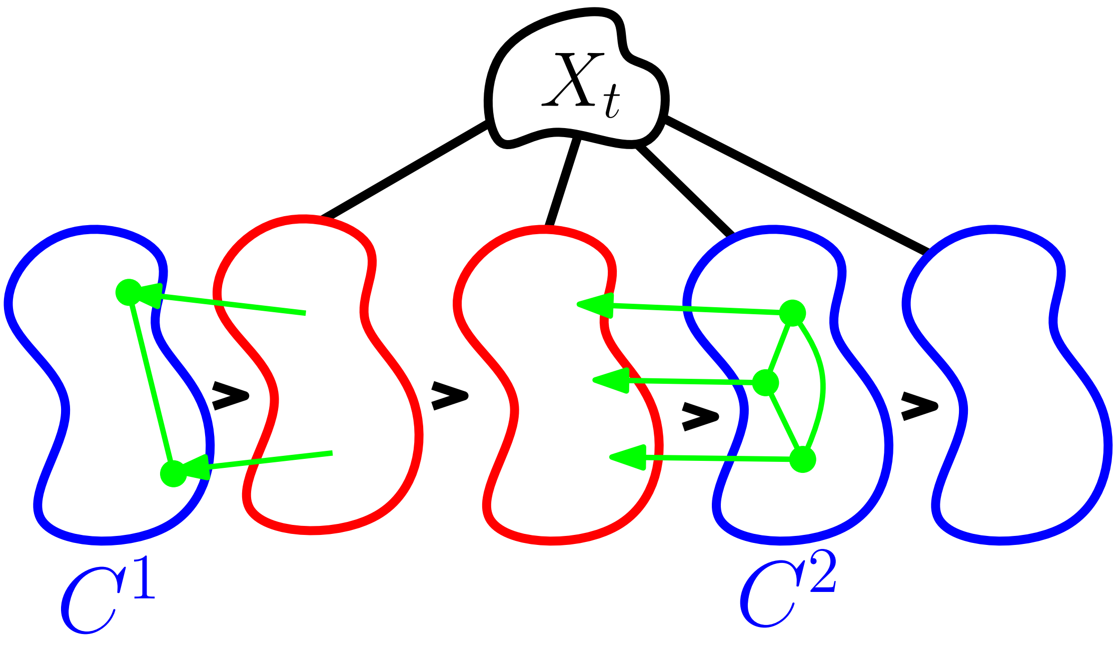


Figure 4.8: The case when there is only one block of the bag X_t. The cover guards can be only in C^1 and C^2. Furthermore, in each of the two components they cannot be comparable, hence they form a clique in both C^1 and C^2.

in at least one bag of a tree decomposition, and so we arrive at a contradiction with $\mathrm{tw}(I(\mathcal{P})) \leq k$. Hence we conclude that $|\mathcal{A}_t| \leq 2k+2$ and the claim holds. $\Diamond$

Finally, consider a node t which has at least 2 blocks. By Property **3** of Lemma 4.17, it holds that t has a neighbor t' in T_t^r such that $X_{t'} = X_t \setminus \{v\}$ and t' has 1 block. By Claim 4.20 we know that $\mathcal{A}_{t'} \leq 2k+2$. Since $L(t) = L(t')$ and $R(t) \subsetneq R(t')$, it follows that $\mathcal{A}_t \subseteq \mathcal{A}_{t'}$, and in particular $|\mathcal{A}_t| \leq |\mathcal{A}_{t'}|$. We have now proved the desired bound for all nodes in $\mathcal{T}$. $\square$

With Lemma 4.18 and Lemma 4.19, we have the tools necessary for arguing that there exists a tree decomposition of the combined graph of small width.

Lemma 4.21. *Let $\mathcal{T} = (T, \mathcal{X})$ be a blocked tree decomposition of $I(\mathcal{P})$ such that $\mathrm{tw}(\mathcal{T}) \leq k$. Then there exists a tree decomposition $\mathcal{T}'$ of $I_C(\mathcal{P})$ of width at most $3k+2$.*

Proof. Consider the tree decomposition $\mathcal{T}' = (T, \mathcal{X}')$ where $\mathcal{X}' = \{X'_t \mid t \in T\}$ is defined as follows. For each $t \in T$ such that its unique neighbor s in T_t^r satisfies $|X_t \setminus X_s| = 1$, we set $X'_t = X_t \cup \mathcal{A}_s$; it will be useful to observe that $\mathcal{A}_s \supseteq \mathcal{A}_t$. For all other nodes $t \in T$, we then set $X'_t = X_t \cup \mathcal{A}_t$. We call nodes of the first type *non-standard* and nodes of the second type standard.

First, we note that the size of each bag in $\mathcal{T}'$ is at most $3k+2$, since every node $t \in T$ satisfies $|\mathcal{A}_t| \leq 2k+2$ by Lemma 4.19. Furthermore, $\mathcal{T}'$ satisfies condition (T1) because $\mathcal{T}$ was a tree decomposition of $I(\mathcal{P})$. $\mathcal{T}'$ also satisfies condition (T2); indeed, for each $v \in \mathcal{P}$ it holds that X'^{v} restricted to standard nodes is a connected tree by Lemma 4.18, and by construction every non-standard node t such that $v \in X'_t \setminus X_t$ is adjacent to a standard node containing v. So, it only remains to argue condition (T3).

Obviously, condition (T3) holds for any edge of $I(\mathcal{P})$. So, consider two elements u, v of $\mathcal{P}$ such that $u \lhd^{\mathcal{P}} v$ or $v \lhd^{\mathcal{P}} u$. If there exists a node $t \in T$ such that $u, v \in X_t$, then $u, v \in X'_t$ and the condition also holds for this edge in $I_C(\mathcal{P})$. So, assume that X^v and X^u are disjoint and let Q be the unique X^v-X^u path in T. By Property **4**, the X^v-X^u path Q in T must contain at least one internal node.

Consider the case where one of these subtrees, say w.l.o.g. X^v, lies in the connected component T_t^r of $T - Q$. Then for each internal node $q \in Q$, it holds that $v \in L(q)$ and $u \in R(q)$, which in turn implies that $v \in \mathcal{A}_q$. Let q_u be the endpoint of Q in X^u and let q_0 be the neighbor of q_u in Q. By Property **4** we have $X_{q_u} \setminus X_{q_0} = \{u\}$, which implies that q_u is a non-standard node and in particular $\mathcal{A}_{q_0} \subseteq X'_{q_u}$. Since q_0 is an internal node of Q, it follows that $v \in X'_{q_u}$ which means that condition (T3) also holds for any edge uv in this case.

Finally, consider the case where there exists a node $q \in Q$ of degree at least 3 such that each of X^u, X^v and r occur in different components of $T - q$. Then we reach a contradiction similarly as in the proof of Lemma 4.18. In particular, since $u, v \in R(q)$ due to the location of the root and there is a cover edge between them, it follows that either u, v occur in the same connected component of X_q or in two consecutive ones, but in either case u, v must lie in the same block of q, say block B. But since $u, v \notin X_q$, this contradicts Property **2** in Lemma 4.17; indeed, each tree in $T - q$ contains at most one of v, u in its bags, and hence there exists no tree T' in $T - q$ satisfying $\bigcup_{t' \in T'} X_{t'} \setminus X_q = B$. Hence this case in fact violates our assumptions and cannot occur.

Summarizing the above arguments, we conclude that each bag in $\mathcal{T}'$ has size at most $3k+2$ and that $\mathcal{T}'$ satisfies all of the conditions of a tree decomposition. ☐

Corollary 4.22. *Let $\mathcal{P}$ be a poset such that* $\mathrm{tw}(I(\mathcal{P})) \leq k$. *Then* $\mathrm{tw}(I_C(\mathcal{P})) \leq 3k+2$.

Proof. By Corollary 3.12 we know that there exists a nice path decomposition of $I(\mathcal{P})$ of width at most k. By Lemma 4.17, it follows that there exists a blocked tree decomposition of $I(\mathcal{P})$ of width at most k. The corollary then follows by Lemma 4.21. ☐

4.2.2 MSO Formulation

In this subsection, we use Fact 3.4 to prove the following result, which forms the second ingredient required for our proof of Theorem 4.13.

Lemma 4.23. #LinExt *is fixed-parameter tractable parameterized by the treewidth of the combined graph of the input poset.*

Proof. Let $\mathcal{P} := (P, \leq^P)$ be a poset. Let G be an (edge-)labeled directed graph obtained from $I_C(\mathcal{P})$ by directing every bidirectional edge of $I_C(\mathcal{P})$, i.e., every edge of $I(\mathcal{P})$, in an arbitrary way and labeling it with the label $\|$.

For a set of edges $E \subseteq E(G)$ with label $\|$, let $G[E]$ be the graph obtained from G after reversing every edge in E. Moreover, for a linear extension $\preceq$ of $\mathcal{P}$ let $E_G(\preceq)$ be the set of edges (u, v) of G such that $v \preceq u$. Note that because every linear extension of $\mathcal{P}$ has to respect the direction of the edges in G given by C, it holds that every edge in $E_G(\preceq)$ has label $\|$.

Claim 4.24. *$E_G(\preceq)$ defines a bijection between the set of linear extensions of $\mathcal{P}$ and the set of subsets E of edges of G with label $\|$ such that $G[E]$ is acyclic.*

Proof of the Claim. Let $\preceq$ be a linear extension of $\mathcal{P}$. Then, as observed above, $E_G(\preceq)$ is a set of edges of G with label $\|$. Moreover, because $G[E_G(\preceq)]$ is a subgraph of $P_G(\preceq)$ and $P_G(\preceq)$ is acyclic so is $G[E_G(\preceq)]$. Hence, $E_G(\preceq)$ is a function from the set of linear extensions of $\mathcal{P}$ to the set of subsets E of edges of G with label $\|$ such that $G[E]$ is acyclic. Towards showing that $E_G(\preceq)$ is injective, assume for a contradiction that this is not the case, i.e., there are two distinct linear extensions $\preceq_1$ and $\preceq_2$ of $\mathcal{P}$ such that $E_G(\preceq_1) = E_G(\preceq_2)$ and let u and v be two elements of $\mathcal{P}$ ordered differently by $\preceq_1$ and $\preceq_2$. Then $\{u, v\} \in I(\mathcal{P})$ and hence either $(u, v) \in E(G)$ or $(v, u) \in E(G)$. The label of (u, v) or (v, u), respectively, is $\|$. W.l.o.g., assume that $(u, v) \in G$ with label $\|$. But then, because $\preceq_1$ and $\preceq_2$ differ on u and v, either $(u, v) \in E_G(\preceq_1)$ but not $(u, v) \in E_G(\preceq_2)$ or $(u, v) \in E_G(\preceq_2)$ but not $(u, v) \in E_G(\preceq_1)$. In both cases we get a contradiction to our assumption that $E_G(\preceq_1) = E_G(\preceq_2)$.

It remains to show that $E_G(\preceq)$ is surjective. To see this let E be a subsets of the edges of G with label $\|$ such that $G[E]$ is acyclic. Because $G[E]$ is acyclic it has a topological ordering, say $\preceq$, of its vertices. Because $G[E]$ contains $C(\mathcal{P})$ as a subgraph and any topological ordering of $C(\mathcal{P})$ is a linear extension of $\mathcal{P}$, we obtain that $\preceq$ is a linear extension and also $E = E_G(\preceq)$. $\Diamond$

It follows from the above that instead of counting the number of linear extensions of $\mathcal{P}$ directly, we can count the number of subsets E of the edges of G with label $\|$ such that $G[E]$ is acyclic. We will show next that there is an MSO_2 formula $\Phi(X)$, whose length is independent of G and can hence be considered constant, such that $G \models \Phi(X)$ if and only if X is a subset of the edges of G with label $\|$ such that $G[E]$ is acyclic. Because of Fact 3.4, this implies that #LinExt is fixed-parameter tractable when parameterized by $\text{tw}(G)$ and hence also when parameterized by $\text{tw}(I_C(\mathcal{P}))$, concluding the proof of the lemma.

Informally, $\Phi(X)$ will check that X is a set of edges of G with label $\|$ and there is no non-empty set of edges C of $G[X]$ that forms a cycle. For the definition of $\Phi(X)$ we will need the following auxiliary formulas.

- The formula $\mathit{edgesin}(X) := \forall x Xx \rightarrow P_{\|}x$, which holds if and only if X is a set of edges of G with label $\|$.
- The formula $\mathit{edgesne}(C) := \exists c Cc \land \forall c Cc \rightarrow Ec$, which holds if and only if C is a non-empty set of edges of G.
- The formula $\mathit{cyclic}(C, X) := \forall v Vv \rightarrow \mathit{degree}(C, X, v)$, which holds if and only if the set C of edges of $G[X]$ contains a cycle.
- The formula $\mathit{degree}(C, X, v) := \mathit{degree}_0(C, X, v) \lor \mathit{degree}_2(C, X, v)$, which holds if and only if either no edge in C is adjacent to v or there are exactly two edges in C that are incident to v such that one of them corresponds to an edge in $G[E]$ with tail v and the other corresponds to an edge in $G[E]$ with head v.
- The formula $\mathit{degree}_0(C, X, v) := \neg\exists c Cc \rightarrow Ivc$, which holds if and only if no edge in C is adjacent to v.
- The formula

  $$\begin{aligned}\mathit{degree}_2(C, X, v) :=& \exists c_i \exists c_o C c_i \land C c_o \land \mathit{in}(X, c_i, v) \land \mathit{out}(X, c_o, v) \land \\ & \forall c (Cc \land \neg c = c_i \land \neg c = c_o) \rightarrow \neg(Hvc \lor Tvc)\end{aligned}$$

  which holds if and only if there are exactly two edges in C that are incident to v such that one of them corresponds to an edge in $G[E]$ with tail v and the other corresponds to an edge in $G[E]$ with head v.
- The formula

  $$\mathit{in}(X, c, v) := (\neg P_{\|}c \land Hvc) \lor (P_{\|}c \land \neg Xc \land Hvc) \lor (P_{\|}c \land Xc \land Tvc)$$

  which holds if and only if v is the head of the edge of $G[X]$ represented by c.
- The formula

  $$\mathit{out}(X, c, v) := (\neg P_{\|}c \land Tvc) \lor (P_{\|}c \land \neg Xc \land Tvc) \lor (P_{\|}c \land Xc \land Hvc)$$

  which holds if and only if v is the tail of the edge of $G[X]$ represented by c.

Then $\Phi(X)$ is the formula:

$$\Phi(X) := \mathit{edgesin}(X) \land \neg\left(\exists C\, \mathit{edgesne}(C) \land \mathit{cyclic}(C, X)\right). \qquad \square$$

Proof of Theorem 4.13. Let $\mathcal{P}$ be the input poset and let $k = \mathrm{tw}(I(\mathcal{P}))$. Then $\mathrm{tw}(I_C(\mathcal{P})) \leq 3k + 2$ by Corollary 4.22, and the theorem follows by Lemma 4.23. $\square$

4.3 Summary and Open Questions

We have given the first parameterized intractability result for counting linear extensions. We hope that the employed techniques will inspire similar results and expand our knowledge about the parameterized complexity of counting problems. In particular, even for #LinExt there remain many open questions concerning other very natural parameterizations such as the width of the poset or the treewidth of the poset graph. Moreover, our intractability result for the treewidth of the cover graph poses the question whether there are stronger parameterizations under which #LinExt becomes tractable, e.g., the treewidth of the poset graph, the treedepth or even vertex cover number of the poset- or cover graph, as well as combinations of these parameters with parameters such as the width, the dimension, or the height of the poset. These numerous examples illustrate that the parameterized complexity of #LinExt is still largely unexplored. As a side note it would also be interesting to establish whether our hardness result for #LinExt can be sharpened to #W[1]-hardness [85] and to obtain matching membership results.

Notes

The results in this chapter appeared in a conference paper in the proceedings of The 24th Annual European Symposium on Algorithms (ESA 2016) [70].

CHAPTER 5

Decomposition Parameters for QBF: Mind the Prefix!

After showing that QBF remains PSPACE-complete even on instances of bounded treewidth [15], Atserias and Oliva introduced a width parameter based on treewidth called *respectful treewidth* which takes into account dependencies between the variables in a QBF formula. They showed that QBF is fixed-parameter tractable parameterized by respectful treewidth provided that a corresponding tree decomposition is given as part of the input. Similar results using almost the same parameter have been obtained for first-order model checking [4] and quantified constraint satisfaction [41]. Other structural parameters such as backdoors have also been studied in the context of QBF [183]. Unfortunately, there still remained fairly simple QBF instances which could not be efficiently solved by any of these approaches (for instance, QBF instances whose graph representation is a star).

Here we develop *prefix pathwidth*, which is a novel decomposition-based parameter that allows an FPT algorithm for QBF. Prefix pathwidth is an extension of pathwidth which takes into account not only the structure of clauses in the formula, but also the structure contained in the quantification of variables. To achieve the latter, we make use of the *dependency schemes* introduced by Samer and Szeider [183, 186], see also the work of Biere and Lonsing [151]. Dependency schemes capture how the assignment of individual variables in a QBF depends on other variables, and research in this direction has uncovered a large number of distinct dependency schemes. The most basic dependency scheme is called the *trivial dependency scheme* [183], which stipulates that each variable depends on all variables with distinct quantification, which come before it in the prefix. Prefix pathwidth uses a completely different approach than previous notions to solve QBF instances, and can in fact be used to efficiently evaluate quantified formulas that remained beyond the reach of state-of-the-art techniques.

Results

We start by introducing our new parameter Section 5.2, where we also show that respectful treewidth is incomparable to prefix pathwidth. Informally, there are two main differences between respectful treewidth and prefix pathwidth: (1) whereas respectful treewidth requires the ordering in which the variables are introduced to be compatible with the dependencies, prefix pathwidth needs the ordering in which the variables are forgotten to be compatible with the dependencies and (2) respectful treewidth is solely defined for the trivial dependency scheme, while prefix pathwidth allows the use of arbitrary permutation dependency schemes.

Afterwards, we focus on developing FPT algorithms for our new measure. When using the trivial dependency scheme, we obtain (by combining Theorem 5.2 with Theorem 5.37):

Theorem 5.1. *QBF is FPT parameterized by the prefix pathwidth with respect to the trivial dependency scheme.*

However, prefix pathwidth can be used in conjunction with any permutation dependency scheme [187], which holds for almost all known dependency schemes; this is reflected in all of our technical results, where we do not fix any particular dependency scheme. In practice, using different dependency schemes may lead to better prefix path decompositions, in turn resulting in significantly faster algorithms.

In their full generality, our main results on solving QBF using prefix pathwidth can be separated into two steps:

1. using a prefix path decomposition of small prefix pathwidth to solve the given QBF I, and
2. finding a suitable prefix path decomposition to be used for step 1.

We resolve the first task by applying advanced dynamic programming techniques on partial existential strategies for the Hintikka game (see e.g. the work of Grädel et al. ([108])) played on the QBF. Essentially, the game approach allows us to translate the question of whether a QBF is true to the question of whether there exists a winning strategy for one player in the Hintikka game. We show that although the number of such strategies is unbounded, at each point in the prefix path decomposition there is only a small number of partial strategies on the processed vertices that need to be considered. Thus we obtain:

Theorem 5.2. QBF *is FPT parameterized by the width of a prefix path decomposition w.r.t. any permutation dependency scheme, when such a decomposition is provided as part of the input.*

Resolving step 2 boils down to a graph-algorithmic problem which is related to the problem of computing various established parameters of directed graphs, such as directed pathwidth or directed treewidth. It is an important open problem whether computing these parameters is FPT or not [191] and the same obstacles seem to also be present for computing our parameter in the general sense. To bypass this barrier, we develop new algorithmic techniques to obtain three distinct algorithms for computing prefix path decompositions. The first of these algorithms, presented in Theorem 5.37, works for the trivial dependency poset as well as other posets which have a similar "layered" structure. The latter two of our algorithms then focus on general posets, but their performance depends on the *poset-width* (i.e., the size of a maximum anti-chain) of the dependency relation; on a high level, the poset-width captures the density of dependencies between variables. In particular, we obtain one polynomial-time approximation algorithm (Theorem 5.35) and one FPT algorithm (Theorem 5.34). In combination with the previous Theorem 5.2, Theorem 5.35 yields one of our main contribution, formalized in Theorem 5.3 below. Observe that here we do not require a decomposition to be part of the input.

Theorem 5.3. *Let τ be a fixed permutation dependency scheme. There exists an FPT algorithm which takes as input a* QBF *I and decides whether I is true in time $f(k,w) \cdot |I|^{\mathcal{O}(1)}$, where f is a computable function, k is the prefix pathwidth and w is the poset-width of I w.r.t. τ.*

Organization of the Chapter

The chapter is organized as follows. We start by formally introducing the QBFs together with dependency schemes and respectful treewidth in Section 5.1. Afterwards in Section 5.2, we formally define our novel parameter — prefix pathwidth — and compare it to respectful treewidth. In Section 5.3, we develop the machinery for efficiently solving QBFs, when a prefix pathwidth decomposition of small with is provided. Finally, in Section 5.4 we develop efficient algorithms for finding suitable prefix pathwidth decompositions for trivial dependency poset as well as in case when the width of dependency poset is bounded.

5.1 Quantified Boolean Formulas

A quantified Boolean formula is a tuple (ϕ, τ) where ϕ is a CNF formula and τ is a sequence of quantified variables, denoted $\text{var}(\tau)$, which satisfies $\text{var}(\tau) \supseteq \text{var}(\phi)$; then ϕ is called the *matrix* and τ is called the *prefix*. A QBF (ϕ, τ) is true if the formula $\tau\phi$ is true. An *assignment* is a mapping from (a subset of) the variables to $\{0, 1\}$.

Given a QBF $I = (\phi, \tau)$ and a partial assignment $\omega : Q \to \{0, 1\}$ where $Q \subseteq \text{var}(\phi)$, we denote by I_ω the subinstance obtained by applying the partial assignment ω; similarly, for a clause $c \in \phi$ we let c_ω denote the clause obtained from c by applying ω.

The primal graph of a QBF $I = (\phi, \tau)$ is the graph G_I defined as follows. The vertex set of G_I consists of every variable which occurs in ϕ, and st is an edge in G_I iff there exists a clause in ϕ containing both s and t.

5.1.1 Dependency Schemes and Posets for QBF

We use *dependency posets* to provide a general and formal way of speaking about the various *dependency schemes* introduced for QBF [183].

We begin by formally defining dependency schemes. For a binary relation $\mathcal{R}$ over some set V we write $\bar{\mathcal{R}}$ to denote its inverse , i.e., $\bar{\mathcal{R}} = \{(y,x) : (x,y) \in \mathcal{R}\}$, and we write $\mathcal{R}^*$ to denote the reflexive and transitive closure of $\mathcal{R}$ i.e., the smallest set $\mathcal{R}^*$ such that $\mathcal{R}^* = \mathcal{R} \cup \{(x,x) : x \in V\} \cup \{(x,y) : \exists z \text{ such that } (x,z) \in \mathcal{R}^* \text{ and } (z,y) \in \mathcal{R}\}$. Moreover, we let $\mathcal{R}(x) = \{y : (x,y) \in \mathcal{R}\}$ for $x \in V$ and $\mathcal{R}(X) = \bigcup_{x\in X} \mathcal{R}(x)$ for $X \subseteq V$. Given a QBF $I = (\phi, \tau)$ we will also need the following binary relation over $\text{var}(\tau)$: $\mathcal{R}_I = \{\, (x,y) \mid x,y \in \text{var}(\tau),\ x \text{ is before } y \text{ in the prefix}\,\}$.

To define dependency schemes we need also the notion of *shifting*, which takes some subset of variables of QBF I in prefix and puts them together with their quantification, in the same relative order, to the end (*down-shifting*) or to the beginning (*up-shifting*) of the prefix.

Definition 5.4 (Shifting [183])**.** Let $I = (\phi, \tau)$ be a QBF and $A \subseteq \text{var}(\tau)$. We say that $I' = (\phi, \tau')$ is obtained from I by *down-shifting* (*up-shifting*) A, in symbols $I' = S^{\downarrow}(I, A)$ ($I' = S^{\uparrow}(I, A)$), if I' is obtained from I by reordering quantifiers in the prefix such that the following holds:

1. $A = \mathcal{R}_{I'}(x)$ ($A = \overline{\mathcal{R}_{I'}}(x)$) for some $x \in \text{var}(\tau)$ and
2. $(x,y) \in \mathcal{R}_{I'}$ iff $(x,y) \in \mathcal{R}_I$ for all $x,y \in A$ and
3. $(x,y) \in \mathcal{R}_{I'}$ iff $(x,y) \in \mathcal{R}_I$ for all $x,y \in \text{var}(\tau) \setminus A$

Definition 5.5 (Dependency Scheme [183])**.** A dependency scheme D assigns to each QBF I a binary relation $D_I \subseteq R_I$ such that I and $S^{\downarrow}(I, D_I^*(x))$ are equivalent for every variable x of I.

It is important to note that dependency schemes in general are too broad a notion for our purposes. Namely, for our algorithm for QBF using prefix pathwidth, we require dependency schemes that lead to satisfiability equivalent QBF instances even after several shifting operations. This is why we will focus our attention on so-called permutation dependency schemes [187], which are known to satisfy this condition.

Definition 5.6 (Permutation [187])**.** A dependency scheme D is a *permutation dependency scheme* if for every QBF $I = (\phi, \tau)$ and every linear extension D' of D^*, it holds that the QBF obtained by permuting the prefix τ according to D' is equivalent with I.

Note that the definition above implies that permutation dependency schemes preserve satisfiability after several shifting operations because each shifting operation leads to an ordering of the prefix that is a linear extension of the original dependency scheme.

Given a QBF $I = (\phi, \tau)$ and a permutation dependency scheme D, a dependency poset $\mathcal{V} = (\mathrm{var}(\phi), \leq^I)$ of I is a poset over $\mathrm{var}(\phi)$ with the following properties:

1. $x \leq^I y$ iff $(x, y) \in D_I^*$ for all $x, y \in \mathrm{var}(\phi)$ and
2. for every linear extension $<$ of $\mathcal{V}$, it holds that I and the QBF instance obtained from I after reordering its prefix according to $<$ are equivalent.

The trivial dependency scheme assigns to each variable x the closest variables on the right of x with different quantification. This gives rise to the trivial dependency poset, which has a certain "layered" structure; more details about these posets are presented in Subsection 5.4.2. However, more refined dependency posets are known to exist and can be computed efficiently [183].

To illustrate these definitions, consider the following QBF I:

$$\forall x \exists y \forall u \exists v (x \vee \neg y \vee v) \wedge (\neg u \vee \neg v \vee y) \wedge (\neg x \vee u \vee \neg v).$$

As an example, consider the following dependency poset on variables of I: $x \leq^I u \leq^I v$, and y is incomparable to all other variables. Up-shifting of the downward-closed set $\{x, u\}$ yields the QBF I':

$$\forall x \forall u \exists y \exists v (x \vee \neg y \vee v) \wedge (\neg u \vee \neg v \vee y) \wedge (\neg x \vee u \vee \neg v).$$

One can readily see that I and I' are both true. The trivial dependency poset over I is the poset given by the chain $x \leq^I y \leq^I u \leq^I v$, where every downward-closed set cannot be further up-shifted.

5.1.2 Respectful Treewidth

For our comparison to prefix pathwidth, we need to define respectful treewidth as it has for instance been introduced in [41, 4, 15].

We start by giving an alternative definition of treewidth in terms of so-called elimination orderings as this characterization is more suited to our definition of respectful treewidth.

Definition 5.7 (Elimination ordering)**.** An elimination ordering of a graph is a linear order of its vertices. Given an elimination ordering ϕ of the graph G, the fill-in graph H of G w.r.t. ϕ is the unique minimal graph such that:

- $V(G) = V(H)$.
- $E(H) \supseteq E(G)$.
- If $0 \leq k < i < j \leq n$ and $v_i, v_j \in N_H(v_k)$, then $v_i v_j \in E(H)$.

The *width* of elimination ordering ϕ is the maximum number of neighbors of any vertex v that are larger than v (w.r.t. ϕ) in H.

The following proposition shows that tree decompositions and elimination orderings are two equivalent characterizations of treewidth.

Proposition 5.8 ([141])**.** *Let G be a graph. The following three claims are equivalent:*

- *G has treewidth k,*
- *G has an elimination ordering of width k,*

Let $I = (\phi, \tau)$ be an QBF instance. An elimination ordering of G_I is *respectful* if it is a linear extension of the reverse of the trivial dependency scheme for I; intuitively, this corresponds to being forced to eliminate variables that have the most dependencies first. The respectful treewidth is then defined as the minimum width of any respectful elimination ordering of G_I.

5.2 Prefix Pathwidth for QBF

Let $G = (V, E)$ be a graph and $\leq^V$ be a partial order of V. For a vertex $v \in V$, we denote by $D_{\leq^V}(v)$ the downward closure of v w.r.t. $\leq^V$, i.e., the set $\{\, u \in V(G) \mid u \leq^V v \,\}$. Similarly, for $W \subseteq V$ we let $D_{\leq^V}(W) = \bigcup_{v \in W} D_{\leq^V}(v)$.

Let $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be a tree decomposition of G. For a node t of T we denote by T_t the subtree of T with t as a root, by $T_{\leq t}$ the set $\bigcup_{s \in T_t} X_s$, and by $T_{<t}$ the set $T_{\leq t} \setminus X_t$. For a vertex $v \in V(G)$ we denote by $f_{\mathcal{T}}(v)$ the unique node t satisfying $v \in X_t$ and $v \notin X_s$, where s is the parent of t in T. For a path decomposition $\mathcal{P} = (P_1, \dots, P_n)$ of G we define P_i, $P_{\leq i}$, $P_{<i}$, and $f_{\mathcal{P}}(v)$ analogously.

A prefix tree decomposition of $G = (V, E)$ w.r.t. $\leq^V$ is a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ that has the downward closure property, i.e., for every vertex $v \in V$ it holds that $D_{\leq^V}(v) \subseteq T_{\leq f_{\mathcal{T}}(v)}$. Analogously, a prefix path decomposition of $G = (V, E)$ w.r.t. $\leq^V$ is a path decomposition $\mathcal{P}$ that has the *downward closure property.* The prefix treewidth of G w.r.t. $\leq^V$, denoted by $otw_{\leq^V}(G)$, is then the minimum width over all prefix tree decompositions of G. The prefix pathwidth, denoted by $opw(G)$, is then defined analogously.

We note that using the same technique as for path decomposition, one can show that every prefix path decomposition of G can be turned into a nice prefix path decomposition of the same width in polynomial time.

The following theorem shows us that if the width of the dependency poset is small, then prefix pathwidth is actually a good approximation of the prefix treewidth w.r.t. the same dependency poset and hence by using the simpler path decompositions we can get the same result.

Theorem 5.9. *Let $G = (V, E)$ be a graph and w the width of the poset $(V, \leq^V)$. Then $opw_{\leq^V}(G) \leq w \cdot otw_{\leq^V}(G)$.*

Proof. Let $\mathcal{T} = (T, \{X_t\}_{t \in T})$ be an optimal prefix tree decomposition of G w.r.t. $\leq^V$. We begin our proof by showing the following claim:

Claim 5.10. *Let W be a chain of the poset $(V, \leq^V)$. Then there exists a leaf-to-root path in T that contains each $f_{\mathcal{T}}(v)$ for every $v \in W$.*

Proof of the Claim. Suppose for a contradiction that the set $\{ f_{\mathcal{T}}(v) \mid \text{for all } v \in W \}$ does not lie on a path. This means that there exist two vertices $u, v \in W$ such that $f_{\mathcal{T}}(v) \notin T_{f_{\mathcal{T}}(u)}$ and $f_{\mathcal{T}}(u) \notin T_{f_{\mathcal{T}}(v)}$. W.l.o.g., we can assume that $u \leq^V v$. The downward closure property of $\mathcal{T}$ then implies that $u \in T_{\leq f_{\mathcal{T}}(v)}$, but then either $u \in X_{f_{\mathcal{T}}(v)}$ and $f_{\mathcal{T}}(v) \in T_{f_{\mathcal{T}}(u)}$, or $u \notin X_{f_{\mathcal{T}}(v)}$ and by then the downward closure property also $f_{\mathcal{T}}(u) \in T_{f_{\mathcal{T}}(v)}$, in either case leading to a contradiction. ◊

From the above claim it follows that if $\mathcal{T}$ does not contain unnecessary nodes, i.e., a node t of $\mathcal{T}$ is unnecessary if $X_t \subseteq X_p$ for the parent p of t in $\mathcal{T}$, then $\mathcal{T}$ has at most w leaves. Hence, the prefix path decomposition $\mathcal{P} := (P_1, \dots, P_h)$, where h is the height of T, defined by $P_i = \{ v \mid v \in B, \text{ where } B \text{ is a bag of distance } i \text{ from the root of } T \}$ is a prefix path decomposition of G w.r.t. $\leq^V$ of width at most w times the width of $\mathcal{T}$. □

We build on the above definitions to define the notions we need on QBFs. A prefix path decomposition of a QBF $I = (\phi, \tau)$ w.r.t. a dependency poset $\mathcal{V} = (\text{var}(\phi), \leq^I)$ is a prefix path decomposition of the primal graph G_I w.r.t. $\leq^I$. The prefix pathwidth of I is then the minimum width over all prefix path decompositions of G_I w.r.t. $\mathcal{V}$.

5.2.1 Comparison to Respectful Treewidth

Respectful treewidth is based on Q-resolution and thus decomposes the dependency structure *beginning from variables that have the most dependencies* (i.e., may appear last in the prefix). Proposition 5.11 shows that both approaches are, in principle, incomparable: there exist classes of QBF instances where one approach leads to polynomial-time algorithms and the other does not, and vice-versa. In contrast, our parameter prefix pathwidth is based on bounding the number of viable strategies in the classical two-player game characterization of the QBF problem. As such, it decomposes the dependency structure of a QBF instance *beginning from variables that have the least dependencies* (i.e., may appear earlier in the prefix). Proposition 5.11 shows that both approaches are, in principle, incomparable: there exist classes of QBF instances where one approach leads to polynomial-time algorithms and the other does not, and vice-versa.

Lemma 5.11. *Let us fix the trivial dependency poset. There exist infinite classes $\mathcal{A}, \mathcal{B}$ of QBF instances such that:*

a. $\mathcal{A}$ has unbounded respectful treewidth but prefix pathwidth at most 1*;*

b. $\mathcal{B}$ has unbounded prefix pathwidth (and prefix treewidth) but respectful treewidth at most 1*.*

Proof. **a.** Let

$$A_i = \exists x_1, \dots, x_i \forall y \exists x (y \vee x) \wedge \bigwedge_{j=1}^{i} (x_j \vee x).$$

The trivial dependency poset $\mathcal{P}_i$ for A_i would be $\{x_1, \dots, x_i\} \leq y \leq x$. Hence every respectful elimination ordering must start with x, and then the width of such an ordering would be $i + 1$. On the other hand, it is straightforward to verify that the path decomposition $\mathcal{Q} = (Q_1, \dots, Q_{i+1})$, where $Q_j = \{x_j, x\}$ for $1 \leq j \leq i$ and $Q_{i+1} = \{y, x\}$, is a prefix path decomposition w.r.t. $\mathcal{P}_i$ of width 1.

b. Consider the following formula with alternating prefix:

$$B_i = \exists x_1 \forall x_2 \dots \forall x_{2^i} \exists x_{2^i+1} \bigwedge_{j=1}^{2^{i-1}} ((x_j \vee x_{2j}) \wedge (x_j \vee x_{2j+1}).$$

Since the quantifiers in the prefix of B_i alternate, the trivial dependency poset $\mathcal{P}_i$ for B_i would be the linear order $x_1 \leq x_2 \leq \dots \leq x_{2^i}$. It is readily observed that the primal graph of B_i is a balanced binary tree of depth i, and it is known that the pathwidth of such trees is $i - 1$ [58]. From the fact that pathwidth is a trivial lower bound for prefix pathwidth together with Theorem 5.9, it follows that $i - 1$ is a lower bound on the prefix treewidth of B_i.

On the other hand, since $\mathcal{P}_i$ is linear order, the only respectful elimination ordering is the reverse of $\mathcal{P}_i$. Moreover, from the definition of B_i, it is easily seen that x_j has at most 1 neighbor that is smaller w.r.t. $\mathcal{P}_i$, namely $x_{\lfloor j/2 \rfloor}$. Therefore, the respectful treewidth of B_i is one. $\square$

Before we continue with the main technical part of this Chapter, we provide one specific example illustrating that our parameter cover some natural classes of instances not covered by previous approaches. Recall that the *vertex cover number* of G is the minimum size of a vertex cover in G. The vertex cover number has often been used as a structural parameter for graph problems which do not have FPT algorithms parameterized by treewidth (see for instance [79]).

Theorem 5.12. QBF *is fixed-parameter tractable parameterized by the vertex cover number of the primal graph.*

Proof. Let $I = (\phi, \tau)$ be an instance of QBF and let k be the vertex cover number of G_I. Let $n = |\text{var}(\tau)|$ and let Z be a vertex cover of G_I of cardinality k; recall that a vertex cover of cardinality k can be computed in time at most $\mathcal{O}(2^k \cdot n)$ [64]. For each

$i \in [1, n]$, let v_i be the i-th variable in the prefix of I, and let $P_i = Z \cup \{v_i\}$. Now consider $\mathcal{P} = (P_1, \ldots, P_n)$. We claim that $\mathcal{P}$ is a prefix path decomposition of width at most k.

Indeed, clearly each P_i in $\mathcal{P}$ contains at most $k+1$ vertices. It is straightforward to verify that conditions (T1) and (T2) hold, and condition (T3) must also hold since every edge has at most one endpoint outside of Z. Finally, $\mathcal{P}$ also has the downward closure property, since for each vertex v_i the set $D_{\leq^V}(v_i)$ is a subset of $Z \cup \{\, v_j \mid j < i \,\}$, and hence $D_{\leq^V}(v) \subseteq P_{\leq f_{\mathcal{P}}(v)}$.

It follows that I has prefix pathwidth at most k. To conclude the proof, we merely need to use the construction above to obtain a prefix path decomposition of small width and then apply the main result of the following section, namely Theorem 5.27. $\square$

5.3 Using Prefix Pathwidth

In this section we will show that deciding the satisfiability of a QBF is fixed-parameter tractable parameterized by the width of a prefix path decomposition which is assumed to be provided as part of the input. The next section will then show how such a prefix path decomposition can be computed efficiently.

5.3.1 Section Overview

The route to the main goal of this section, i.e., an FPT algorithm for QBF, can be conceptually separated into three parts, each corresponding to one subsection. First, our techniques essentially rely on the well-known Hintikka Games [108], which we introduce in the next subsection. In particular, the notion of a "winning existential strategy" will be crucial for the algorithm; a QBF instance is true if and only if the existential player has a winning strategy. Second, we show that even though the number of existential strategies can be potentially unbounded, they can be grouped into a small (i.e., bounded by k) number of equivalence classes. This equivalence is formalized in Definition 5.18 via the use of so-called "signatures". The final subsection then presents the dynamic programming algorithm itself; the algorithm maintains and dynamically computes records of relevant signatures, which contain all the needed information about existential strategies on the dynamically processed variables.

5.3.2 Hintikka Games

Alternating prenex form For the definition of Hintikka Games (and in particular their strategies), it will be convenient to use an equivalent but more structured representation of QBFs. A QBF is in *alternating prenex form* if the prefix has the form $\forall y_1 \exists x_1, \ldots, \forall y_\ell \exists x_\ell$. Any QBF in alternating prenex form can then be represented as a tuple (ϕ, Y, X) where ϕ is the matrix and $Y = (y_1, \ldots, y_\ell)$ and $X = (x_1, \ldots, x_\ell)$ are disjoint ordered sets of the variables in the prefix.

We remark that any QBF can be transformed into alternating prenex form in linear time by the addition of dummy variables, i.e., variables which do not occur in the matrix. It is readily observed that if two dummy variables occur consecutively in the prefix, then they can both be deleted without changing the truth value of the QBF. As a consequence, we may freely assume that the number of dummy variables will never be greater than $2 \cdot |\text{var}(\phi)| + 1$. Moreover, adding the dummy variables does not change the prefix path decomposition since the dummy variables do not occur in the matrix but solely in the prefix of the QBF formula. In the remainder of this section, we will assume that every QBF is in alternating prenex form.

Hintikka Games Given a QBF (ϕ, Y, X) such that $|X| = |Y| = \ell$, a *strategy for Eloise* (an *existential strategy*) is a sequence of mappings $\mathcal{T} = (\tau_i : \{0,1\}^i \to \{0,1\})_{i=1,\ldots,\ell}$. An existential strategy $\mathcal{T}$ is *winning* if, for any mapping $\delta : \{y_1, \ldots, y_n\} \to \{0,1\}$, the formula ϕ is true under the assignment $y_i \mapsto \delta(y_i)$ and $x_i \mapsto \tau_i(\delta(y_1), \ldots, \delta(y_i))$ for $1 \le i \le \ell$. A *partial existential strategy* is a sequence of mappings $\mathcal{T} = (\tau_i : \{0,1\}^i \to \{0,1\})_{i=1,\ldots,\ell'}$, for some $\ell' \le \ell$.

A *strategy for Abelard* (a *universal strategy*) is defined analogously, whereas the mappings δ and τ are swapped, and we call a universal strategy winning if ϕ is not true. Formally, it is a sequence of mappings $\Lambda = (\lambda_i : \{0,1\}^{i-1} \to \{0,1\})_{i=1,\ldots,\ell}$. A universal strategy Λ is *winning* if, for any mapping $\delta : \{x_1, \ldots, x_n\} \to \{0,1\}$, the formula ϕ is false under the assignment $x_i \mapsto \delta(x_i)$ for $1 \le i \le \ell$, $y_1 \mapsto \lambda_1$, and $y_i \mapsto \lambda_i(\delta(x_1), \ldots, \delta(x_{i-1}))$ for $2 \le i \le \ell$.

A mapping δ from a subset of Y to $\{0,1\}$ is called a *universal play*, and similarly a mapping δ from a subset of X to $\{0,1\}$ is called an *existential play*. It will sometimes be useful to view plays as binary strings, and in this context we will use the symbol $\circ$ to denote the concatenation of two strings; for instance, if $\delta(x_1) = 1$ and $\delta(x_2) = 0$, then one can represent δ as $(1,0)$, and $(1,0) \circ (0) = (1,0,0)$. It is easily observed that plays on dummy variables do not need to be taken into account by a winning existential strategy.

Proposition 5.13 (folklore)**.** *A QBF I is true iff there exists a winning existential strategy on I iff there exists no winning universal strategy on I.*

Let α be a partial existential strategy restricted to $X' = (x_1, \ldots, x_a)$ and let β be a universal play over $Y' = \{y_1, \ldots, y_b\}$. Then the pair (β, α) results in a partial assignment δ of $X' \cup Y'$, formally given as follows (for i up to $\min(a,b)$): $\delta(y_i) = \beta(y_i)$ and $\delta(x_i) = \alpha(\beta(y_1), \beta(y_2), \ldots, \beta(y_i))$. We denote this as $(\beta, \alpha) \rightsquigarrow \delta$. For brevity, we also sometimes just write (β, α) for the assignment δ given by $(\beta, \alpha) \rightsquigarrow \delta$.

For the remainder of this section, we fix the following notions. Let $I = (\phi, Y, X)$ be a QBF, let $\le^I$ be a partial order forming a dependency poset of I (w.r.t. some permutation dependency scheme), and let $\mathcal{P} := (P_1, \ldots, P_n)$ be a prefix path decomposition of I w.r.t. $\le^I$ of width k. Moreover, for every i with $1 \le i \le n$, let $B_i = P_i$ be a bag in $\mathcal{P}$, $D_i = D_{\le^I}(P_{<i})$, $C_i = P_{<i}$ (see Figure 5.1), and let I be up-shifted on D.

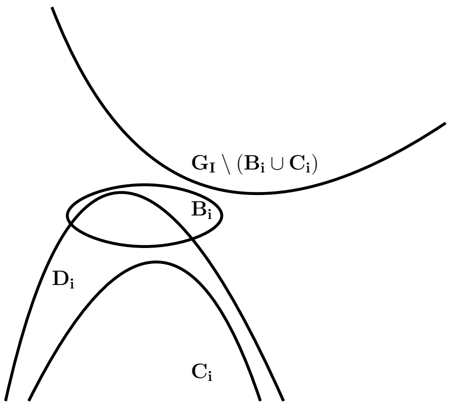


Figure 5.1: B_i is a bag in $\mathcal{P}$ that separates C_i, i.e., vertices forgotten in some bag before B_i, from the rest of the graph. D_i is the downward closure of C_i w.r.t. $\leq^I$.

Observation 5.14. *For any i with $1 \leq i \leq n$, B_i forms a separator in G_I and hence each clause in ϕ either contains only variables in $P_{\leq i}$ or only variables in $(Y \cup X) \setminus C_i$. Furthermore, $D_i \subseteq P_{\leq i}$.*

Hintikka games allow us to decide the truthfulness of a QBF by computing all strategies for the existential player. We will show next that even though the number of possible strategies that can be used for the variables in each $P_{\leq i}$ is huge, it is sufficient to only remember a small number of "representative strategies" that can be used on $P_{\leq i}$ to allow dynamic programming along the prefix path decomposition. The proof of this claim is based on considering two layers of equivalences and showing that they both only have a small number of equivalence classes.

5.3.3 Equivalence of Assignments

The first equivalence, which serves as the building block for the latter one, considers assignments of the variable set D_i.

Definition 5.15. Let δ_1 and δ_2 be two partial assignments of D_i. Then $\delta_1 \approx \delta_2$ iff $I_{\delta_1} = I_{\delta_2}$.

It is readily observed that $\approx$ is an equivalence.

Lemma 5.16. *$\approx$ has at most $2^{2^{\mathcal{O}(k)}}$ equivalence classes.*

Proof. Consider two partial assignments δ_1, δ_2 of D_i. Let U be the set of all possible clauses over $B_i \setminus D_i$ (including the empty clause); clearly $|U| \leq 3^k$. Let U_1 contain the clauses from U which occur in I_{δ_1}, and similarly for U_2 and I_{δ_2}. Let δ_i' be the restriction of δ_i to variables from $D_i \cap B_i$.

Claim 5.17. *If $U_1 = U_2$ and $\delta'_1 = \delta'_2$, then $\delta_1 \approx \delta_2$.*

Proof of the Claim. Clearly, I_{δ_1} and I_{δ_2} are defined on the same variables. It remains to show that both contain the same clauses. Let c be a clause of I. Because of Observation 5.14 either $c \subseteq (X \cup Y) \setminus C_i$ or $c \subseteq P_{\leq i}$. In the first case, it follows from $\delta'_1 = \delta'_2$ that $c_{\delta_1} = c_{\delta_2}$ and hence I_{δ_1} contains c_{δ_1} if and only if so does I_{δ_2}. In the later case, we obtain that $c_{\delta_1} \subseteq B_i \setminus D_i$. Hence, because $U_1 = U_2$, we obtain that I_{δ_1} contains c_{δ_1} if and only if so does I_{δ_2}. $\Diamond$

The Lemma then follows from the above claim because there are at most $2^{|U|} = 2^{3^k}$ possible choices of U_i and at most 2^k possible choices of δ'_i. $\square$

5.3.4 Equivalence of Strategies

For a partial existential strategy α on D_i, we denote by S_α (referred to as the *signature*) the set containing each instance I such that there exists a universal play β which together with α results in I; formally, $S_\alpha = \{ I_\delta \mid \exists \text{ universal play } \beta \text{ such that } (\beta, \alpha) \rightsquigarrow \delta \}$.

Definition 5.18. Let α_1 and α_2 be two partial existential strategies on D_i. Then $\alpha_1 \equiv \alpha_2$ iff $S_{\alpha_1} = S_{\alpha_2}$.

Once again, it is easy to verify that $\equiv$ is transitive, reflexive and symmetric, and hence is an equivalence relation. We show that its index is also upper-bounded by a function of k.

Lemma 5.19. *For any partial existential strategy α on D_i it holds that $|S_\alpha| \leq 2^{2^{\mathcal{O}(k)}}$. Furthermore, $\equiv$ has at most $2^{2^{2^{\mathcal{O}(k)}}}$ equivalence classes.*

Proof. By Lemma 5.16 each partial assignment δ of D_i results in one of at most $2^{2^{\mathcal{O}(k)}}$ many distinct QBF instances I_δ. Because S_α is a subset of the set of all these instances for any partial existential strategy α, it follows that $|S_\alpha| \leq 2^{2^{\mathcal{O}(k)}}$ and moreover that there are at most $2^{2^{2^{\mathcal{O}(k)}}}$ distinct choices for S_α. $\square$

Lemma 5.20. *Let I be a QBF and let α be a winning existential strategy for I. Then, for any partial existential strategy α' which is a subset of α, it holds that I' is true for any $I' \in S_{\alpha'}$.*

Proof. Assume for a contradiction that this is not case and let α' be a defined on the variables in $X' \cup Y'$, where $X' \subseteq X$ and $Y' \subseteq Y$ and let $I' \in S_{\alpha'}$ be a no-instance. Because $I' \in S_{\alpha'}$ there is a universal play β_0 on the variables in Y' such that $I' = I_{\delta_0}$, where $(\beta_0, \alpha') \rightsquigarrow \delta_0$. Because I' is a no-instance there is an universal play β_1 on the variables in $Y \setminus Y'$ such that the instance I_δ, where $(\beta_0 \circ \beta_1, \alpha) \rightsquigarrow \delta$, contains the empty clause. Hence, the universal play $\beta_0 \circ \beta_1$ wins against α on I, contradicting our assumption that α is a winning strategy. $\square$

We will also need the converse of the above claim, formulated below.

Observation 5.21. *If there exists a partial existential strategy α on D_i such that each $I' \in S_\alpha$ is true, then I is true.*

Proof. A winning existential strategy for I can simply apply α until it reaches a true subinstance $I' \in S_\alpha$. From there on, it can continue with a winning existential strategy for I'. □

5.3.5 The Algorithm

In this subsection, we develop a dynamic programming algorithm on a nice prefix path decomposition $\mathcal{P} = (P_1, \dots, P_n)$ of I to decide whether I is true. Recall that by the above, for each partial existential strategy α on D_i there is a signature S_α. For each D_i, we will compute the set K_i of all signatures corresponding to any partial existential strategy on D_i; formally, $K_i = \{ S_\alpha \mid \alpha \text{ is an existential strategy on } D_i \}$. We call K_i the signature set of D_i, and the algorithm proceeds by computing the sets $K_1, \dots, K_n$ for the bags $P_1, \dots P_n$. One key observation is that for the construction of the sets K_i one only needs to consider a special type of partial existential strategies on D_i, which we will call oblivious.

A (partial) existential strategy α on $X_0 = (x_1, \dots, x_j)$ is *oblivious* if it does not distinguish between universal plays that lead to the same reduced instance. More precisely, if two universal plays on the first l universal variables result in the same reduced instance, then an oblivious (partial) existential strategy α shall not distinguish between these two universal plays in the moves following after l. Formally, α is oblivious if it satisfies the following condition for every partial existential strategy α' obtained as a restriction of α to $(x_1, \dots, x_l)$, $l < j$, and for every two universal plays β_1, β_2 on $(y_1, \dots, y_l)$ such that $I_{\delta_1} = I_{\delta_2}$ where $(\beta_1, \alpha') \rightsquigarrow \delta_1$ and $(\beta_2, \alpha') \rightsquigarrow \delta_2$. Let p satisfy $l < p \leq j$, and for each $\beta_p = \{0,1\}^{p-l}$ let $(\beta_1 \circ \beta_p, \alpha) \rightsquigarrow \delta_1''$ and similarly $(\beta_2 \circ \beta_p, \alpha) \rightsquigarrow \delta_2''$. Then, for every x_i where $l < i \leq p$, it holds that $\delta_1''(x_i) = \delta_2''(x_i)$. The following shows we can compute K_i, by merely considering signatures of oblivious partial existential strategies.

Lemma 5.22. *Let I be a* QBF*. For any partial existential strategy there is an oblivious partial existential strategy that has the same signature.*

Proof. Let α be a partial existential strategy of I. If α is oblivious, then the claim of the lemma holds. So assume that α is not oblivious. We will show how to transform α into an oblivious partial existential strategy without changing the signature. Let l be the smallest number such that the restriction α' of α to $(x_1, \dots, x_l)$ violates the definition of obliviousness for some universal plays β_1, β_2 on $(y_1, \dots, y_l)$. Let $S = \{ I_\delta \mid \exists \beta' \in \{0,1\}^l : (\beta', \alpha') \rightsquigarrow \delta \}$, and for each $I_\delta \in S$ let us choose an arbitrary representative β_δ such that $(\beta_\delta, \alpha') \rightsquigarrow \delta$.

Now consider the strategy α'' which copies α in all mappings except for the following. For each universal play β_1 on $(y_1, \dots, y_l)$ such that β_1 is distinct from each of the representatives β_δ, for each p where $l < p$, and for each $\beta_p \in \{0,1\}^{p-l}$, we set $\alpha''_p(\beta_1 \circ \beta_p) := \alpha_p(\beta_\delta \circ \beta_p)$, where δ satisfies $I_\delta = I_{(\beta_1,\alpha)}$. Observe that α'' no longer violates the condition of obliviousness for any β_1 on $(y_1, \dots, y_l)$. Additionally, if the condition of obliviousness was satisfied by α for a pair of universal plays β'_1, β'_2 then it remains satisfied also by α''. For now, assume α'' has the same signature as α; then by repeating the above procedure until we obtain an oblivious strategy would proof the claim of the lemma.

To complete the proof, we argue that α'' has the same signature as α. Let β be an assignment of Y partitioned into β_1 (on $(y_1, \dots, y_l)$) and β_r (on the remaining universal variables) and let δ_1 be the assignment obtained as $(\beta_1, \alpha'') \rightsquigarrow \delta_1$. Moreover, let β_δ be the representative of I_{δ_1} and let δ_2 be the assignment obtained as $(\beta_\delta, \alpha) \rightsquigarrow \delta_2$. Then, $I_{\delta_1} = I_{\delta_2}$ and $(\beta_1 \circ \beta_r, \alpha'')$ is equal to $(\beta_\delta \circ \beta_r, \alpha)$ on all variables x_i, y_i with $i > l$ (by definition of α''). Hence, $I_{\delta'_1} = I_{\delta'_2}$, where $(\beta_1 \circ \beta_r, \alpha'') \rightsquigarrow \delta'_1$ and $(\beta_\delta \circ \beta_r, \alpha) \rightsquigarrow \delta'_2$, as required. □

The algorithm then consists of the following four procedures, each tied to a specific claim:

1. *Initialization*(Observation 5.23): this is the procedure that is called at the beginning of the algorithm, i.e., for the empty bag P_1.
2. *Introduce* (Observation 5.24): this is the procedure that is called whenever we have computed K_{i-1} and P_i is an introduce node.
3. *Forget* (Lemma 5.25): this is the procedure that is called whenever we have computed K_{i-1} and P_i is a forget node.
4. *Termination* (Observation 5.26): this is the procedure that is called when we have computed K_n.

The claims are provided below. We remark that each procedure not only computes the next signature set, but also implicitly ensures that I is up-shifted on D_i.

Observation 5.23. *There exists a constant-time algorithm which takes as input a QBF instance I and a prefix path decomposition $\mathcal{P}$ and outputs K_1.*

Proof. Since D_1 is empty, K_1 contains only a single signature (the signature of the empty strategy), which contains I. In other words, $K_1 = \{\{I\}\}$. Observe that I is up-shifted on D_1. □

Observation 5.24. *There exists a constant-time algorithm which takes as input a QBF instance I, a prefix path decomposition $\mathcal{P}$ and the signature set K_{i-1} and outputs K_i when P_i is an introduce node.*

Proof. Assume $P_i = P_{i-1} \cup \{z\}$, where $z \in X \cup Y$. Then $P_{<i-1} = P_{<i}$ and in particular $z \notin D_i$. Hence $K_i = K_{i-1}$. Observe that if I was up-shifted on D_{i-1}, then I will also be up-shifted on D_i. $\square$

Lemma 5.25. *There exists an FPT algorithm which takes as input a QBF I, a prefix path decomposition $\mathcal{P}$ such that I is up-shifted on D_{i-1} and the signature set K_{i-1} and outputs K_i when P_i is a forget node.*

Proof. Assume $P_i = P_{i-1} \setminus \{z\}$, where $z \in X \cup Y$. Then $z \in P_{<i}$ but $z \notin P_{<i-1}$, which implies that $D_i = D_{i-1} \cup \mathcal{D}_{\leq^I}(z)$. The algorithm checks whether $z \in D_{i-1}$ or not. If $z \in D_{i-1}$, then $\mathcal{D}_{\leq^I}(z) \subseteq D_{i-1}$ and hence $D_i = D_{i-1}$. This means that $K_i = K_{i-1}$.

If $z \notin D_{i-1}$, then let $Z = \mathcal{D}_{\leq^I}(z) \setminus D_{i-1} = D_i \setminus D_{i-1}$. Observe that $Z \subseteq P_i$ and hence $|Z| \leq k$. We apply up-shifting on D_i; since I was already up-shifted on D_{i-1}, this means that after up-shifting the prefix of I will contain first the variables in D_{i-1}, followed by the variables in Z, and then all remaining variables. Our goal is now to expand the signature set K_{i-1} by considering all possible results of an existential strategy and universal play on Z. We first formalize the notion of extended signature below, and then describe how the algorithm proceeds.

Let S be a signature in K_{i-1} and let $I_\delta \in S$. Let $\mathcal{A}$ be the set of all partial existential strategies in I_δ on the variable set $Z \cap X$. Since Z forms a prefix of I_δ and $|Z| \leq k$, it follows that $|\mathcal{A}| \leq 2^{2^{\mathcal{O}(k)}}$. Similarly, let $\mathcal{B}$ be the set of all universal plays in I_δ on the variable set $Z \cap Y$, and observe $|\mathcal{B}| \leq 2^k$. The extended signature w.r.t. I_δ and $\alpha_0 \in \mathcal{A}$ is the set $S^{I_\delta}_{\alpha_0} = \{\, I_\omega \mid \omega = \delta \cup \delta',\ \text{whereas } \exists \beta_0 \in \mathcal{B} : (\beta_0, \alpha_0) \rightsquigarrow \delta' \text{ within } I_\delta \,\}$. Observe that, by the bound on $|\mathcal{B}|$, it follows that each extended signature can be computed from a given I_δ and α_0 in $2^k \cdot |\phi|$ time.

The algorithm begins by setting $K'_i := \emptyset$ and iteratively processes each $S \in K_{i-1}$ as follows. Let $S = \{I_1, \dots, I_m\}$. The algorithm branches over all m-tuples of (possibly non-distinct) partial existential strategies from $\mathcal{A}$, and for each such $\tau = (\alpha_1, \alpha_2, \dots, \alpha_m) \in \mathcal{A}^m$ it produces m pair-wise extended signatures $S^{I_1}_{\alpha_1}, S^{I_2}_{\alpha_2}, \dots, S^{I_m}_{\alpha_m}$. It then computes their union $S_\tau = \bigcup_{j \in [m]} S^{I_j}_{\alpha_j}$, and sets $K'_i := K'_i \cup \{S_\tau\}$. The procedure specified in this paragraph can be carried out in FPT time, since

a) the number of distinct elements in K_i and also in K_{i-1} is bounded by the number of equivalence classes of $\equiv$, which is at most $2^{2^{2^{\mathcal{O}(k)}}}$ by Lemma 5.19, and

b) the cardinality of each $S \in K_{i-1}$ is at most $2^{2^{\mathcal{O}(k)}}$, also by Lemma 5.19.

It remains to show that $K'_i = K_i$. We first show that if $S \in K'_i$, then $S \in K_i$. Because $S \in K'_i$, there exists $S_0 \in K_{i-1}$, where $S_0 = \{I_1, \dots, I_m\}$, along with an m-tuple $\tau := (\alpha_1, \dots, \alpha_m)$ of partial existential strategies such that $S = \bigcup_{j \in [m]} S^{I_j}_{\alpha_j}$. Let us fix one arbitrary partial existential strategy α_0 on D_{i-1} with the signature S_0. Then, α_0

partitions all possible assignments $\mathcal{B}$ of $Y \cap D_{i-1}$ into sets $\mathcal{B}_1, \dots, \mathcal{B}_m$ where $\beta \in \mathcal{B}_j$ iff $(\beta, \alpha_0) \rightsquigarrow I_j$.

Consider the partial existential strategy α on D_i which proceeds as follows. On D_{i-1}, it copies α_0. On $D_i \setminus D_{i-1}$, it takes each universal play β and decomposes it into $\beta_0 \circ \beta_z$, where β_0 is the universal play on D_{i-1} and β_z is the universal play on the remaining variables, i.e., a subset of $D_i \setminus D_{i-1}$ and copies α_j on β_z. By construction, it follows that α has the signature S, which implies that $S \in K_i$, as required.

Now assume that $S \in K_i$. Because of Lemma 5.22, it holds that S is the signature of an oblivious partial existential strategy α on D_i. Let α_0 be the restriction of α to D_{i-1}, and let $S_0 = \{I_1, \dots, I_m\}$ be the signature of α_0 in K_{i-1}. Then α_0 once again partitions all possible assignments $\mathcal{B}$ of $Y \cap D_{i-1}$ into sets $\mathcal{B}_1, \dots, \mathcal{B}_m$ where $\beta \in \mathcal{B}_j$ iff $I_{(\beta,\alpha_0)} = I_j$.

For any $j \in [m]$, let β_j be an arbitrary universal play in $\mathcal{B}_j$. Moreover, let α_j be the partial existential strategy (operating on I_j) defined as follows. For each universal play β_z on (a subset of) $Y \cap Z$, we let α_j copy the move of α against the universal play $\beta_j \circ \beta_z$. Observe that because α is oblivious, α_j is independent of the actual choice of β_j in $\mathcal{B}_j$.

We claim that $S = S_\tau$ for $\tau = (\alpha_1, \dots, \alpha_m)$, which implies that $S \in K_i'$. We first show that if $I \in S$ then $I \in S_\tau$. Hence, let $I \in S$. Then, there is a universal play β on D_i such that $I = I_{(\beta,\alpha)}$. Let β_0 be the restriction of β to $D_{i-1} \cap Y$, let β_1 be the restriction of β to $Z \cap Y$, and let j be such that $\beta_0 \in \mathcal{B}_j$. Then, because α is oblivious, it holds that $I = I_{(\beta_j \circ \beta_1, \alpha)} \in S_{\alpha_j}^{I_j}$. This shows that $I \in S_\tau$, as required.

For the reverse direction let $I \in S_\tau$. Then, there is a $j \in [m]$ such that $I \in S_{\alpha_j}^{I_j}$. Hence, there is a universal play β_z on $Z \cap Y$ such that $I = I_{(\beta_j \circ \beta_z, \alpha)}$, which implies $I \in S$, as required. □

Observation 5.26. *There exists a constant-time algorithm which takes as input a QBF instance I and a prefix path decomposition $\mathcal{P} = \{P_1, \dots, P_n\}$ and the signature set K_n and decides whether I is true.*

Proof. After processing the last bag P_n of $\mathcal{P}$, it holds that $D = X \cup Y$ and hence every signature in K_n can only contain two possible instances: the trivial true instance I_T which contains no variables and no clauses, and the trivial false instance I_F, which contains no variables and the empty clause. If $\{I_T\} \in K_n$, then the algorithm outputs true, and otherwise it outputs false. Correctness follows from Lemma 5.20 and Observation 5.21. □

Having established the procedures for the individual nodes of the path decomposition, we can now prove the correctness of the whole dynamic programming algorithm. We note that the following theorem applies to arbitrary QBF instances (not only those in alternating prenex form).

Theorem 5.27. *There exists an FPT algorithm which takes as input a* QBF *I, an integer parameter k, and a prefix path decomposition $\mathcal{P}$ of I of width at most k and decides whether I is true.*

Proof. If I is not already in alternating prenex form, we convert it by adding dummy variables as described in Subsection 5.3.2. As mentioned in the preliminaries, it is possible to transform a prefix path decomposition $\mathcal{P}$ into a nice prefix path decomposition $\mathcal{P}'$ containing at most $\mathcal{O}(k|I|)$ bags; let $n = |\mathcal{P}'|$. It then computes the signature set K_1 by Observation 5.23 and proceeds by iteratively computing $K_2, K_3, \ldots, K_n$ by Observation 5.24 and Lemma 5.25. Once the algorithm computes the signature set K_n, it outputs based on Observation 5.26. □

5.4 Computing Prefix Pathwidth

This section is devoted to parameterized and approximation algorithms for computing the prefix pathwidth. Observe that if the given partial ordering is empty, then the prefix pathwidth of the graph G is the same as the pathwidth of G. Moreover, the downward closure property is checkable in polynomial. time Thus, computing the prefix pathwidth is NP-complete.

Before we present our algorithms, we will state some interesting observations about prefix path decompositions. For the remainder of this section let G be a graph and $(V(G), \leq^V)$ a poset on $V(G)$ of width w. The first observation relates prefix pathwidth with a well-known decomposition parameter for directed graphs, i.e., directed pathwidth [19].

Definition 5.27.1 (Directed path decomposition ([19]))**.** Let D be a directed graph. A directed path decomposition is a sequence $\mathcal{P} := (P_1, \ldots, P_n)$ of subsets of vertices of D such that:

(D1) $\bigcup_{1 \leq i \leq n} P_i = V(D)$,

(D2) for every $u \in V(D)$, the set $D_u = \{\, i \in \{1, \ldots, n\} \mid u \in W_i \,\}$ induces an interval, and

(D3) for each $uv \in E(D)$ there are i and j with $1 \leq i \leq j \leq n$ such that $u \in W_i$ and $v \in W_j$.

The *width* of a directed path decomposition and the directed pathwidth of D, denoted by $dpw(D)$ are defined analogously to the corresponding notions for path decompositions.

Observation 5.28. *Let D be the directed graph obtained from G by replacing every edge by two anti-parallel arcs and adding an arc uv for every $u, v \in V(G)$ such that $u \leq^V v$. Then, $opw_{\leq^V}(G) = dpw(D)$.*

Proof. We will show an even stronger statement, namely we show that any prefix path decomposition of G w.r.t. $\leq^V$ is also a directed path decomposition of the graph D and vice versa.

Suppose that $\mathcal{P} := (P_1, \ldots, P_n)$ is a prefix path decomposition of G w.r.t. $\leq^V$. Then, $\mathcal{P}$ satisfies Properties (D1) and (D2), because $\mathcal{P}$ satisfies Properties (T1) and (T2).

Towards showing Property (D3), let uv be any arc in D. If uv is also an edge of G, then Property (D3) follows because $\mathcal{P}$ satisfies Property (T3). Otherwise, $u \leq^V v$ and because of the downward closure property of $\mathcal{P}$, we obtain that $u \in P_{\leq f_{\mathcal{P}}(v)}$. Hence, there is an i with $i \leq f_{\mathcal{P}}(v)$ and $u \in P_i$, which because $v \in P_{f_{\mathcal{P}}(v)}$ implies Property (D3).

On the other hand, let $\mathcal{P}$ be a directed path decomposition of D. Then, properties (T1) and (T2) follow immediately from properties (D1) and (D2) of $\mathcal{P}$. Towards showing Property (T3), let $uv \in E(G)$ be an edge of G. Then, both uv and vu are arcs in D and it follows from Property (D3) that there are i, j, i', and j' with $1 \leq i < j \leq n$ and $1 \leq i' \leq j' \leq n$ such that $u \in P_i$, $v \in P_j$, $v \in P_{i'}$, and $u \in P_{j'}$. Because of Property D2, we obtain that $u \in P_l$ for every l between i and j' and $v \in P_l$ for every l between i' and j. Hence, because $i < j$ and $i' < j'$ there exists an h with $1 \leq h \leq n$ such that $u, v \in P_h$, showing Property (T3).

Towards showing the downward closure property for $\mathcal{P}$, recall that for every $u \in D_{\leq^V}(v)$ there exists a directed edge uv in D. It hence follows from Property (D3) that either there exists i with $1 \leq i \leq n$ such that $u, v \in W_i$, or there are i and j with $1 \leq i < j \leq n$ such that $u \in W_i$ and $v \in W_j$. Therefore, $u \in P_{\leq f_{\mathcal{P}}(v)}$ for every $u \in D_{\leq^V}(v)$ and $\mathcal{P}$ has the downward closure property. $\square$

Since it has been shown [191] that deciding whether the directed pathwidth of a digraph is at most k is solvable in polynomial-time for every fixed k, the above observation implies that the same holds for the prefix pathwidth. It is an important open question, however, whether computing directed pathwidth is fixed-parameter tractable.

Let G be a graph and $\leq$ a linear order on $V(G)$. We call the pair $(G, \leq)$ an ordered graph. We say that an ordered graph $(G, \leq)$ is an ordered minor of an ordered graph $(G', \leq')$, if $(G, \leq)$ can be obtained from $(G', \leq')$ by a sequence of operations of any of the following kind: (1) vertex deletion, (2) edge deletion, or (3) ordered edge-contraction, i.e., an edge-contraction for which the resulting vertex can take the place of any of the endpoints of the contracted edge in the ordering $\leq'$. Let X be a set and $\leq$ a reflexive and transitive binary relation on X. Then, X is well-quasi ordered w.r.t. $\leq$ if every infinite sequence $x_1, x_2, \ldots$ of elements of X contains an increasing pair $x_i \leq x_j$ where $i < j$. If the set of ordered graphs of prefix-pathwidth at most k was well-quasi ordered w.r.t. the ordered minor relation, this would be an important step towards a FPT algorithm for determining whether a graph has prefix-pathwidth at most k [134]. However, we show that this is actually not the case.

Observation 5.29. *The set of ordered graphs is not well-quasi ordered w.r.t. the ordered minor relation.*

Proof. It has been shown that the set of all permutations of the natural numbers is not well-quasi ordered w.r.t. the removal of entries [188]. In particular, Bóna and Spielman [188] constructed an infinite antichain of permutations for this setting. We will make use of this antichain to construct an infinite antichain of ordered graphs w.r.t. the ordered minor relation.

Let $\mathcal{P} = (p_1, p_2, \dots)$ be the infinite sequence of permutations (also called an antichain) witnessing that the set of all permutations is not well-quasi ordered w.r.t. to the removal operation as defined by Bóna and Spielman [188]. We define a corresponding sequence $\mathcal{G} = (g_1, g_2, \dots)$ of ordered graphs as follows: for every permutation $p \in \mathcal{P}$, let $g(p)$ be the ordered graph $(G, \leq)$, where G is a path given by the sequence $(v_0, v_1, \dots, v_{|P|})$ of vertices (with endpoints v_0 and $v_{|P|}$) and the partial ordering $\leq$ is defined by $v_i \leq v_j$ if and only if $p^{-1}[i] \leq p^{-1}[j]$ for every $0 < i, j \leq n$. Note that because G is a path and the endpoint v_0 is the only vertex of G, which is not comparable (w.r.t. $\leq$) to any other vertex, it holds that $g(p) = g(p')$ if and only if $p = p'$ (hence g is a bijection). We set $\mathcal{G} := (g(p_1), g(p_2), \dots)$. It remains to show that $\mathcal{G}$ is an infinite antichain w.r.t. to the ordered minor relation, i.e., there is no pair i, j with $i < j$ such that g_i is an ordered minor of g_j.

Suppose for a contradiction that there is such a pair say i, j such that $i < j$ and $g_i := (G_i, \leq_i)$ is an ordered minor of $g_j := (G_j, \leq_j)$. Then, in particular, G_i is a minor of G_j and because both G_i and G_j are paths we can assume, w.l.o.g., that g_i is obtained from g_j by a sequence of ordered edge-contractions. Moreover, because any ordered edge-contraction of an edge $e = \{u, v\} \in G_j$ for which the resulting vertex takes the place of say u in the ordering leads to an ordered graph isomorphic to $g(p'_j)$, where p'_j is the permutation obtained from p_j after removing the element corresponding to v in p_j, we obtain that any permutation p such that $g_i = g(p)$ is can be obtained from any permutation p' such that $g_j = g(p')$. Finally, because $g(p)$ is a bijection, we obtain that p_i can be obtained from p_j via the removal of elements, contradicting our assumption that the set of all permutations is well-quasi ordered under removal of elements. $\square$

The above observations suggest that fixed-parameter tractability for computing prefix pathwidth is a difficult question.

5.4.1 Algorithms for Posets of Bounded Width

In the following we will give two algorithms that compute the prefix path decomposition of a graph that are efficient in the case that the given poset has small width. Our first algorithm shows that if the width of the poset $\leq^V$ is bounded by a constant, then deciding whether G has a prefix path decomposition w.r.t. $\leq^V$ of width at most k is fixed-parameter tractable (in k).

For a subset $V' \subseteq V$, let $P_{\leq^V}(V')$ be the prefix of V' w.r.t. $\leq^V$, i.e., the set of all vertices v in V' with $D_{\leq^V}(v) \subseteq V'$, and let $S_{\leq^V}(V')$ be the suffix of V' w.r.t. $\leq^V$, i.e., the set $V' \setminus P_{\leq^V}(V')$. Finally, we let $\delta(V')$ be the vertices in V' with at least one neighbor in $V \setminus V'$. We call the set $\delta(V')$ guards of V'.

Let $B(V')$ be the bipartite graph with bipartition $(\delta(V'), N(\delta(V')) \setminus V')$ containing all edges of G with one endpoint in $\delta(V')$ and the other endpoint in $N(\delta(V')) \setminus V'$. Moreover, for a prefix path decomposition $\mathcal{P} := (P_1, \dots, P_n)$ of G w.r.t. $\leq^V$ and for every i with $1 \leq i \leq n$, let $B_\mathcal{P}(i)$ be the bipartite graph $B(P_{\leq^V}(P_{\leq i}))$.

Before we proceed to proof the required statements for the algorithm, let us first provide an informal overview of the algorithm. The main observation behind the algorithm (which is shown in Lemma 5.30 below) is that in any prefix path decomposition of G w.r.t. $\leq^V$, the intersection between any two bags, say P_i and P_{i+1}, can be characterized by a pair (D, C), where D is a downward closed set of vertices of G equal to $P_{\leq^V}(P_{\leq i})$ and C is a minimal vertex cover of the bipartite graph between the guards of D and the neighbors of these guards in the remainder of G, i.e., the bipartite graph $B_\mathcal{P}(i)$. Given this crucial observation, it is then straightforward to define simple conditions for deciding whether a pair (D, C) can be the intersection of two bags in a prefix path decomposition of width at most k as well as conditions for deciding whether the intersection of two bags corresponding to a pair (D, C) can be followed by (in some prefix path decomposition of width at most k) the intersection of two bags corresponding to the pair (D', C') (these conditions are defined in Lemma 5.31). Computing a prefix path decomposition then boils down to deciding whether there is a directed path from the pair $(\emptyset, \emptyset)$ to the pair $(V(G), \emptyset)$ in the digraph whose vertex set consists of all pairs (D, C) such that (D, C) can be the intersection between two bags in some prefix path decomposition of width at most k and whose arcs are defined using the above mentioned conditions (see the proof of Theorem 5.34). Because the number of downward closed sets is bounded by $|V(G)|^w$ and one can show (see Lemma 5.33) that the number of possible minimal vertex covers (for each downward closed set) is bounded by k^{2k}, this then leads to the required result.

We are now ready to proof the formal statements required by the algorithm as outlined above.

Lemma 5.30. *Let $\mathcal{P} := (P_1, \dots, P_n)$ be a prefix path decomposition of G w.r.t. $\leq^V$ of width at most k, which is minimal in the sense that no bag of $\mathcal{P}$ contains unnecessary vertices. Then, for every i with $1 \leq i < n$, it holds that $P_i \cap P_{i+1}$ is a minimal vertex cover of $B_\mathcal{P}(i)$.*

Proof. We first show that $P_i \cap P_{i+1}$ contains a vertex cover of $B_\mathcal{P}(i)$ for every i with $1 \leq i < n$. Suppose not, then there is an i with $1 \leq i < n$ such that $P_i \cap P_{i+1}$ is not a vertex cover of $B_\mathcal{P}(i)$. Hence, there is an edge $\{v, u\} \in E(B_\mathcal{P}(i)) \subseteq E(G)$ with $v \in P_{\leq^V}(P_{\leq i})$ and $u \in N(\delta(P_{\leq^V}(P_{\leq i}))) \setminus (P_{\leq^V}(P_{\leq i}))$ such that $v, u \notin P_i \cap P_{i+1}$. Hence, either $u \notin P_{\leq i}$ or $u \in S_{\leq^V}(P_{\leq i})$. In the former case, because $v \in P_{\leq i}$ but $v \notin P_{i+1}$, we obtain that property (T3) of $\mathcal{P}$ is violated. In the latter case, because $S_{\leq^V}(P_{\leq i}) \subseteq P_i$, we obtain that $u \notin P_{i+1}$ and hence $f_\mathcal{P}(u) = i$. Consequently, $D_{\leq^V}(u) \nsubseteq P_{\leq i} = P_{\leq f(u)}$, contradicting the downward closure property of $\mathcal{P}$.

Towards showing the minimality of the vertex cover, assume for a contradiction that this is not the case and there is an i with $1 \leq i < n$ such that $P_i \cap P_{i+1}$ is not a minimal vertex cover of $B_\mathcal{P}(i)$. Then, there is a vertex $v \in P_i \cap P_{i+1}$ such that $(P_i \cap P_{i+1}) \setminus \{v\}$ is still a vertex cover of $B_\mathcal{P}(i)$. Then, either $v \in P_{\leq^V}(P_{\leq i})$ or $v \in S_{\leq^V}(P_{\leq i})$.

In the first case, let $\mathcal{P}' := (P'_1, \dots, P'_n)$ be obtained from $\mathcal{P}$ after removing v from every bag P_j with $j \geq i+1$. We show that $\mathcal{P}'$ is still a prefix path decomposition, contradicting our assumption that no bag of $\mathcal{P}$ contained unnecessary vertices. It is easy to verify that conditions (T1) and (T2) as well as the downward closure property still hold for $\mathcal{P}'$. Towards showing (T3), let $\{v, u\} \in E(G)$. Because $(P_i \cap P_{i+1}) \setminus \{v\}$ is a vertex cover of $B_{\mathcal{P}}(i)$, we obtain that if $\{v, u\} \in B_{\mathcal{P}}(i)$, then $u \in P'_i$, as required. If $\{v, u\} \notin B_{\mathcal{P}}(i)$, then $u \in P_{\leq^V}(P_{\leq i})$ and because $\mathcal{P}$ satisfies condition (T3), it follows that $\{u, v\}$ is covered by some bag P_j with $j \leq i$, as required.

In the second case, let $\mathcal{P}' := (P'_1, \dots, P'_n)$ be obtained from $\mathcal{P}$ after removing v from every bag P_j with $j \leq i$. We show that $\mathcal{P}'$ is still a prefix path decomposition, contradicting our assumption that no bag of $\mathcal{P}$ contained unnecessary vertices. It is easy to verify that conditions (T1) and (T2) as well as the downward closure property still hold for $\mathcal{P}'$. Towards showing (T3), let $\{v, u\} \in E(G)$. Because $(P_i \cap P_{i+1}) \setminus \{v\}$ is a vertex cover of $B_{\mathcal{P}}(i)$, we obtain that if $\{v, u\} \in B_{\mathcal{P}}(i)$, then $u \in P'_{i+1}$, as required. If $\{v, u\} \notin B_{\mathcal{P}}(i)$, then either $u \in S_{\leq^V}(P_{\leq i})$ in which case because of the downward closure property of $\mathcal{P}$ also $u \in P_{i+1}$ and hence $u \in P'_{i+1}$, as required, or $u \in V(G) \setminus P_{\leq i}$ and because $\mathcal{P}$ satisfies condition (T3), we obtain that $\{v, u\}$ is covered by some bag P_j with $j \geq i+1$, as required. □

Lemma 5.31. *There is a prefix path decomposition of G w.r.t. $\leq^V$ of width at most k if and only if there is a sequence $\mathcal{S} := ((D_0, C_0), \dots, (D_n, C_n))$ of pairs (D_i, C_i) such that:*

C1 $D_0 = C_0 = \emptyset$, $D_n = V(G)$, $C_n = \emptyset$,

C2 for every i with $0 \leq i \leq n$:

C2A D_i is downward closed,

C2B C_i is a minimal vertex cover of $B(D_i)$,

C3 for every i with $0 \leq i < n$:

C3A $|(D_{i+1} \setminus D_i) \cup C_i \cup C_{i+1}| \leq k$,

C3B $C_i \setminus C_{i+1} \subseteq D_{i+1}$,

C3C $D_i \subseteq D_{i+1}$.

Proof. For the forward direction, suppose there is a prefix path decomposition $\mathcal{P} := (P_1, \dots, P_n)$ of G w.r.t. $\leq^V$ of width at most k. W.l.o.g., we can assume that $\mathcal{P}$ is minimal (in the sense that no bag contains unnecessary vertices). It hence follows from Lemma 5.30 that $P_i \cap P_{i+1}$ is a minimal vertex cover of $B_{\mathcal{P}}(i)$ for every i with $1 \leq i < n$.

We claim that $\mathcal{S} := ((D_0, C_0), (D_1, C_1), \dots, (D_n, C_n))$ with $D_0 := \emptyset$, $C_0 := \emptyset$, $D_i := P_{\leq^V}(P_{\leq i})$, $C_i := P_i \cap P_{i+1}$ for every i with $1 \leq i < n$, $D_n = V(G)$, and $C_n = \emptyset$ is the required sequence. Conditions C1 and C2A are trivially satisfied. Condition C2B follows from Lemma 5.30 applied to $\mathcal{P}$. Towards condition C3A observe that

$$\begin{aligned}
& |(D_{i+1} \setminus D_i) \cup C_i \cup C_{i+1}| \\
= \; & |((P_{\leq^V}(P_{\leq i+1})) \setminus (P_{\leq^V}(P_{\leq i}))) \cup \\
& (P_i \cap P_{i+1}) \cup (P_{i+1} \cap P_{i+2})| \\
\leq \; & |((P_{\leq^V}(P_{\leq i+1})) \setminus (P_{\leq^V}(P_{\leq i}))) \cup P_{i+1}| \\
\leq \; & |((P_{\leq i+1}) \setminus (P_{\leq^V}(P_{\leq i}))) \cup P_{i+1}| \\
\leq \; & |P_{i+1} \cup P_{i+1}| \\
\leq \; & k
\end{aligned}$$

The second to last inequality follows because $P_{\leq i+1} \setminus P_{\leq^V}(P_{\leq i}) \subseteq P_{i+1}$, which in turn can be seen as follows. Let $v \in P_{\leq i+1} \setminus P_{\leq^V}(P_{\leq i})$, then either $v \in P_{\leq i+1} \setminus P_{\leq i}$ or $v \in S_{\leq^V}(P_{\leq i})$. In the former case $v \in P_{i+1}$ (due to the definition of $P_{\leq i+1}$) and in the later case $v \in P_{i+1}$ because of the downward closure property of $\mathcal{P}$.

Towards showing C3B, it suffices to show that $v \in C_i \setminus C_{i+1}$ implies $v \in D_{i+1}$ for every $v \in V(G)$. Because $C_i \setminus C_{i+1} = (P_i \cap P_{i+1}) \setminus (P_{i+1} \cap P_{i+2}) = (P_i \cap P_{i+1}) \setminus P_{i+2}$, we obtain that $v \in C_i \setminus C_{i+1}$ implies that $f_{\mathcal{P}}(v) = i + 1$. Hence, using the fact that $\mathcal{P}$ satisfies the downward closure property, we obtain that $D_{\leq^V}(v) \subseteq D_{i+1}$. In particular, $v \in D_{i+1}$, as required.

Condition C3C, i.e., $D_i \subseteq D_{i+1}$ or equivalently $P_{\leq^V}(P_{\leq i}) \subseteq P_{\leq^V}(P_{\leq i+1})$ is satisfied because $P_{\leq i} \subseteq P_{\leq i+1}$ and the downward closed subset of a set contains the downward closed subset of any subset of the set.

For the reverse direction suppose that we are given $\mathcal{S} := ((D_0, C_0), \ldots, (D_n, C_n))$ satisfying the conditions given in the statement of the lemma. We claim that $\mathcal{P} := (P_1, \ldots, P_n)$ with $P_i := D_i \setminus D_{i-1} \cup C_{i-1} \cup C_i$ for every i with $1 \leq i \leq n$ is a prefix path decomposition of G w.r.t. $\leq^V$ of width at most k. Because of condition C1, it holds that $P_{\leq i} = \bigcup_{1 \leq j \leq i} D_j \setminus D_{j-1} \cup C_{j-1} \cup C_j = \bigcup_{1 \leq j \leq i} D_j \cup C_j$ and again using condition C1, we obtain $P_{\leq n} = V(G)$. It follows that $\mathcal{P}$ satisfies property T1.

To show the remaining properties for $\mathcal{P}$, we need the following claim.

Claim 5.32. *Let $v \in V(G)$ be any vertex such that $v \in P_i \setminus P_{i+1}$, then $v \in D_i$, $v \notin C_i$, and all neighbors of v outside of D_i are in C_i.*

Proof of the Claim. Let $v \in V(G)$ be any vertex such that $v \in P_i \setminus P_{i+1} = (D_i \setminus D_{i-1} \cup C_{i-1} \cup C_i) \setminus (D_{i+1} \setminus D_i \cup C_i \cup C_{i+1})$ for some i with $1 \leq i < n$, i.e., v is forgotten at position i. There are two cases for this to happen, i.e., either $v \in D_i \setminus D_{i-1}$ or $v \in C_{i-1}$. Note that in both cases $v \notin C_i$. In the latter case, we obtain from C3B that $v \in D_i$. Hence, we have already shown that $v \in D_i$ and $v \notin C_i$. Moreover, because $v \notin C_i$ we obtain from C2B that C_i contains all the neighbors of v outside of D_i. This completes the proof of the above claim. ◊

To show property (T2) for $\mathcal{P}$, it suffices to show that for any $v \in V(G)$ with $v \in P_i \setminus P_{i+1}$ for some position i with $1 \leq i < n$, it holds that $v \notin P_j = D_j \setminus D_{j-1} \cup C_{j-1} \cup C_j$ for any $j > i+1$. From Claim 5.32, we obtain that $v \in D_i$, $v \notin C_i$, and all the neighbors of v outside of D_i are in C_i. Because $v \in D_i$, we obtain from C3C that $v \notin D_j \setminus D_{j-1}$. Consequently, it is sufficient to show that $v \notin C_j$ for any $j > i+1$. Because C_i contains all the neighbors of v outside of D_i, we obtain from C3B that for every $j > i+1$ it holds that C_j contains all the neighbors of v outside of D_j. Using C2B (in particular the minimality of C_j), we obtain that C_j does not contain v, as required.

We will show property (T3) for $\mathcal{P}$ by showing (by induction on the index of the forgotten vertices) that whenever a vertex is forgotten at position i, then every edge incident to it is contained in some bag $j \leq i$. This clearly holds if v is the first vertex that is forgotten, because all neighbors of v in D_i must still be in P_i and all neighbors of v outside of D_i are in $C_i \subseteq P_i$ (because of Claim 5.32). So assume that v is the l-th vertex that is forgotten and v is forgotten at some position i. Again, all neighbors of v in D_i are either still in P_i or have been forgotten before v and hence (by the induction hypotheses) every edge incident to the forgotten neighbors is contained in some bag P_j with $j \leq i$. Moreover, because of Claim 5.32 all the neighbors of v outside of D_i are in C_i and hence also in P_i, as required.

We show next that $\mathcal{P}$ has the downward closure property. Again, because of Claim 5.32, if a vertex v is forgotten at some position i, then $v \in D_i$. The downward closure property for $\mathcal{P}$, then follows from C2A.

Because of C3A, we have that the width of $\mathcal{P}$ is at most k. Hence, $\mathcal{P}$ is a prefix path decomposition of G w.r.t. $\leq^V$ of width at most k, as required. □

Lemma 5.33 (Damaschke [55], Theorem 4.). *Let $D \subseteq V(G)$ be a downward closed set w.r.t. $\leq^V$, then there are at most 2^k minimal vertex covers of $B(D)$ of size at most k and these can be enumerated in time $\mathcal{O}(|E(B(D))| + k^2 2^k)$.*

In the following let $w = \text{width}((V(G), \leq^V)$ and let $(W_1, \dots, W_w)$ be a chain partition of $(V(G), \leq^V)$ (which due to Proposition 3.10 can be computed in polynomial-time). For a vector $v \in [0, |W_1|] \times \dots \times [0, |W_w|]$, let $D_{\leq^V}(v)$ be the set of vertices of $V(G)$ that contains the first $v[i]$ vertices from every chain W_i. Note that for every downward closed set V' there is a vector $v \in [0, |W_1|] \times \dots \times [0, |W_w|]$ such that $V' = D_{\leq^V}(v)$, which we denote by $v_{\leq^V}(V')$.

Let $\mathcal{P} := (P_1, \dots, P_n)$ be a prefix path decomposition of G w.r.t. $\leq^V$ of width at most k. Note that because $P_{\leq^V}(P_{\leq i})$ is downward closed the vector $v(P_{\leq^V}(P_{\leq i}))$ is well-defined.

Theorem 5.34. *There is an algorithm that finds a prefix path decomposition of G w.r.t. $\leq^V$ of width at most k or decides that no such prefix path decomposition exists in time $\mathcal{O}((|V(G)|^w 2^k)^2 |V(G)|)$. Hence, for any constant w computing a prefix path decomposition is fixed-parameter tractable parameterized by k.*

Proof. To decide whether G has a prefix path decomposition w.r.t. $\leq^V$ of width at most k, we first build an auxiliary directed graph H as follows.

The vertex set of H consists of all pairs (D, C) such that $D \subseteq V(G)$ is a downward closed set and C is a minimal vertex cover of $B(D)$ of size at most k. Furthermore, there is an arc from (D, C) to (D', C') of H if and only if (D, C) and (D', C') satisfy the conditions C3A-C3C given in Lemma 5.31 (with $D_i = D$, $C_i = C$, $D_{i+1} = D'$, and $C_{i+1} = C'$). This completes the construction of H. Because of Lemma 5.31, we obtain that G has a prefix path decomposition w.r.t. $\leq^V$ of width at most k if and only if there is a directed path in H from $(\emptyset, \emptyset)$ to $(V(G), \emptyset)$. Hence, given H we can decide whether G has a prefix path decomposition w.r.t. $\leq^V$ of width at most k (and output such a path decomposition if it exists) in time $\mathcal{O}(|V(H)| \log(|V(H)|) + E(H))$ (e.g., by using Dijkstra's algorithm). It remains to analyze the time required to construct H (as well as its size).

Because every downward closed set D can be characterized by a vector $v \in [0, |W_1|] \times \ldots \times [0, |W_w|]$ (namely by the vector $v(D)$), there are at most $|V(G)|^w$ such sets D. Moreover, due to Lemma 5.33 for each such D there are at most 2^k minimal vertex covers of $B(D)$ of size at most k. Hence, H has at most $\mathcal{O}(|V(G)|^w 2^k)$ vertices and again using Lemma 5.33, the vertex set of H can be constructed in time $O(|V(G)|^{w+2} k^2 2^k)$. To compute the arc set of H, we go over all of the at most $\mathcal{O}((|V(G)|^w 2^k)^2)$ pairs of vertices of H and check the conditions C3A–C3C, which can be done in time $\mathcal{O}(|V(G)|)$ (for each of these pairs). Hence, the total time required to construct H is $\mathcal{O}((|V(G)|^w 2^k)^2 |V(G)|)$ and since this dominates the time required to find a shortest path in H, this is also the total running time of the algorithm. □

Theorem 5.35. *There is a polynomial-time algorithm that outputs a prefix path-decomposition of G w.r.t. $\leq^V$ of width at most $2w(2k^2 + k) + 1$ or outputs correctly that no prefix path decomposition of G w.r.t. $\leq^V$ of width at most k exists.*

Proof. For a vector $v \in [0, |W_1|] \times \ldots \times [0, |W_w|]$ and i with $1 \leq i \leq w$, let $A(v, i)$ be the set of vertices $\delta(D_{\leq^V}(v)) \cap W_i$, let $B(v, i)$ be the set of vertices $N(\delta(D_{\leq^V}(v)) \cap W_i) \setminus D_{\leq^V}(v))$, and let $G(v, i)$ be the bipartite graph with bipartition $(A(v, i), B(v, i))$ containing all edges of G between $A(v, i)$ and $B(v, i)$. Let $H(G(v, i))$ be the set of vertices of $G(v, i)$ with degree larger than k and let $C(G(v, i))$ consists of $H(G(v, i))$ and all vertices in $A(v, i) \setminus H(G(v, i))$ with at least one neighbor in $G(v, i) \setminus H(G(v, i))$. We also set $P(v)$ to be the set of vertices $\bigcup_{1 \leq i \leq w} C(G(v, i))$.

We are now ready to describe the algorithm that outputs a prefix path decomposition of G w.r.t. to $\leq^V$ of width at most $2w(2k^2 + k) + 1$ or outputs "No" if no prefix path decomposition of G w.r.t. $\leq^V$ of at width at most k exists. The algorithms tries to iteratively construct a sequence $(v_1, \ldots, v_n)$ of vectors $v_i \in [0, |W_1|] \times \ldots \times [0, |W_w|]$ such that v_1 is the all zero vector, v_n is the vector $(|W_1|, \ldots, |W_w|)$, v_{i+1} is obtained from v_i by increasing exactly one component of v_i by one, and $|C(G(v_i, j))| \leq 2k^2 + k$ for every i and j with $1 \leq i \leq n$ and j with $1 \leq j \leq w$. The algorithm tries to find the sequence in a greedy manner, i.e., after having constructed the sequence $(v_1, \ldots, v_i)$

it first checks whether v_i is the vector $(|W_1|, \dots, |W_w|)$ (in which case the algorithm has succeeded). Otherwise it goes over all j with $1 \leq j \leq w$ and checks whether the vector v' obtained from v_i by increasing the j-th entry by one satisfies $v'[j] \leq |W_j|$ and $|C(G(v', j))| \leq 2k^2 + k$. If so it continues on the sequence $(v_1, \dots, v_i, v')$. Otherwise, it outputs that G does not have a prefix path decomposition w.r.t. $\leq^V$ of width at most k.

If the algorithm succeeds, it outputs $\mathcal{P} := (P_1, \dots, P_n)$ with $P_1 = \emptyset$ and $P_i := P(v_i) \cup P(v_{i-1}) \cup (D_{\leq^V}(v_i) \setminus D_{\leq^V}(v_{i-1}))$ for every i with $1 < i \leq n$ as the prefix path decomposition of G w.r.t. $\leq^V$. Otherwise, it returns that G does not have a prefix path decomposition w.r.t. of width at most k. This completes the description of the algorithm.

It is straightforward to check that the algorithm runs in polynomial-time. It remains to show correctness of the algorithm.

We start by showing that if the algorithm succeeds and outputs the sequence $\mathcal{P} := (P_1, \dots, P_n)$, then $\mathcal{P}$ is a prefix path decomposition of G w.r.t. $\leq^V$ of width at most $2w(2k^2 + k) + 1$. Let $(v_1, \dots, v_n)$ be the sequence of vectors computed by the algorithm. Recall that v_1 is the all-zero vector, v_n is the vector $(|W_1|, \dots, |W_n|)$, v_{i+1} is obtained from v_i by increasing exactly one entry by one, and $|C(G(v_i, j)| \leq 2k^2 + k$ for every i and j with $1 \leq i \leq n$ and j with $1 \leq j \leq w$. Because for every i with $1 < i \leq n$, P_i contains (at least) the vertex $D_{\leq^V}(v_i) \setminus D_{\leq^V}(v_{i-1})$, it holds that $\bigcup_{1 \leq i \leq n} P_i = V(G)$, which shows property (T1) for $\mathcal{P}$.

Claim 5.36. *Let $v \in V(G)$ be any vertex such that $v \in P_i \setminus P_{i+1}$, then $v \in D_{\leq^V}(v_i)$, and all neighbors of v outside of $D_{\leq^V}(v_i)$ are in P_i and have degree more than k in $G(v_i, j)$ for every $1 \leq j \leq w$.*

Proof of the Claim. We first show that $v \in D_{\leq^V}(v_i)$. Assume for a contradiction that $v \notin D_{\leq^V}(v_i)$. Then, because $v \in P_i$, it holds that $v \in P(v_i) \cup P(v_{i-1})$. If $v \in P(v_i)$, then $v \in P_{i+1}$ contradicting our assumption. Hence, $v \in P(v_{i-1})$. In particular, there is some j with $1 \leq j \leq w$ such that $v \in B(v_{i-1}, j)$ and either v has degree larger than k in $G(v_{i-1}, j)$. Note that because $v \notin P_{i+1}$, we obtain that $v \notin D_{\leq^V}(v_{i+1}) \setminus D_{\leq^V}(v_i)$ and hence also $v \notin D_{\leq^V}(v_{i+1})$. It follows that $v \in B(v_{i+1}, j)$. Furthermore, if v had degree more than k in $G(v_{i-1}, j)$, then because $v_{i-1} \leq v_{i+1}$, v still has degree more than k in $G(v_{i+1}, j)$ and hence $v \in C(v_{i+1}, j) \subseteq P_{i+1}$ contradicting our assumption. This shows that $v \in D_{\leq^V}(v_i)$.

We show next that all neighbors of v outside of $D_{\leq^V}(v_i)$ are in P_i. Because $v \notin P_{i+1}$, we obtain that $v \notin P(v_i)$. Consequently, for every j with $1 \leq j \leq w$, all neighbors of v in $G(v_i, j)$ have degree more than k. Hence, all neighbors of v outside of $D_{\leq^V}(v_i)$ are in $C(v_i, j) \subseteq P_i$. $\Diamond$

To show property (T2) for $\mathcal{P}$, it suffices to show that for any $v \in V(G)$ with $v \in P_i \setminus P_{i+1}$ for some position i with $1 \leq i < n$, it holds that

$$v \notin P_j = D_{\leq^V}(v_j) \setminus D_{\leq^V}(v_{j-1}) \cup P(v_{j-1}) \cup P(v_j)$$

for any $j > i+1$. From Claim 5.36, we obtain that $v \in D_{\leq^V}(v_i)$, and all the neighbors of v outside of $D_{\leq^V}(v_i)$ have degree more than k in $G(v_i, l)$ for every l with $1 \leq l \leq w$. Because $v \in D_{\leq^V}(v_i)$ and $v_i \leq v_j$, we obtain that $v \notin D_{\leq^V}(v_j) \setminus D_{\leq^V}(v_{j-1})$. Consequently, it is sufficient to show that $v \notin P(v_j)$ for any $j > i+1$. Because all the neighbors of v outside of $D_{\leq^V}(v_i)$ have degree more than k in $G(v_i, l)$ for every l with $1 \leq l \leq w$ and $v_i \leq v_j$, we obtain that for every $j > i+1$ and every l with $1 \leq l \leq w$, it holds that all the neighbors of v outside of $D_{\leq^V}(v_j)$ have degree more than k in $G(v_j, l)$. It follows that $v \notin P(v_j)$, as required.

We will show property (T3) for $\mathcal{P}$ by showing (by induction on the index of the forgotten vertices) that whenever a vertex is forgotten at position i, then every edge incident to it is contained in some bag $j \leq i$. This clearly holds if v is the first vertex that is forgotten, because all neighbors of v in $D_{\leq^V}(v_i)$ must still be in P_i and all neighbors of v outside of $D_{\leq^V}(v_i)$ are in P_i by Claim 5.36. So assume that v is the l-th vertex that is forgotten and v is forgotten at some position i. Again, all neighbors of v in $D_{\leq^V}(v_i)$ are either still in P_i or there have been forgotten before v and hence (by the induction hypotheses) the edges incident to the forgotten neighbors are contained in some bag P_j with $j \leq i$. Moreover, because of Claim 5.36 all the neighbors of v outside of $D_{\leq^V}(v_i)$ are in P_i.

The downward closure property of $\mathcal{P}$ follows immediately from Claim 5.32 and the fact that $D_{\leq^V}(v_i)$ is downward closed for every i with $1 \leq i \leq n$.

By construction the width of $\mathcal{P}$ is at most $2w(2k^2+k)+1$. This shows that $\mathcal{P}$ is a prefix path decomposition of G w.r.t. $\leq^V$ of width at most $2w(2k^2+k)+1$.

We show next that if the algorithm is not successful, then G does not have a prefix path decomposition w.r.t. $\leq^V$ of width at most k. Because the algorithm is not successful, there is a vector $v \in [0, |W_1|] \times \ldots \times [0, |W_w|]$ with $v \neq (|W_1|, \ldots, |W_w|)$ such that for every j with $1 \leq j \leq w$ either $v'[j] > |W_j|$ or $|C(G(v', j))| > 2k^2 + k$, where v' is the vector obtained from v by increasing the j-th entry of v by one. Assume for a contradiction that there is a prefix path decomposition $\mathcal{P} := (P_1, \ldots, P_n)$ of G w.r.t. $\leq^V$ of width at most k. W.l.o.g., we will assume that $|P_i \triangle P_{i+1}| \leq 1$. Let i be the smallest integer with $1 \leq i \leq n$ such that there is a l with $1 \leq l \leq w$ with $v_P[l] > v[l]$, where $v_P := v(P_{\leq^V}(P_{\leq i}))$. Observe that because $|P_i \triangle P_{i+1}| \leq 1$, the integer l is unique and moreover $v_P[l] \leq v[l] + k$. Hence, $v_P[l'] \leq v[l']$ for any $l' \neq l$ and $1 \leq l' \leq w$ and $v[l] < v_P[l] \leq v[l] + k$. It follows that there is a set D of at most k vertices of $G(v, l)$ in $B(v, l)$ such that $G(v, l) \setminus D$ is an induced subgraph of $G(v_P, l)$. Because every vertex in D can reduce the size of $C(v, l)$ by at most k, i.e., if its degree in $G(v, l)$ is exactly k, we obtain that $|C(v_P, l)| \geq |C(v, l)| - k^2 > 2k^2 + k - k^2 = k^2 + k$.

Because of Lemma 5.30, P_i has to contain a vertex cover of $B_{\mathcal{P}}(i)$. In particular, because $G(v_P, l)$ is a subgraph of $B_{\mathcal{P}}(i)$, P_i has to contain a vertex cover of $G(v_P, l)$. It remains to show that because $|C(v_P, l)| > k^2 + k$, $G(v_P, l)$ cannot have a vertex cover of size at most k. Assume for a contradiction that $G(v_P, l)$ has a vertex cover C of size at most k. Clearly, $H(v_P, l) \subseteq C$. Furthermore, because every vertex in $G(v_P, l) \setminus H(v_P, l)$ has degree at most k, any such vertex can cover the edges of at most k non-isolated vertices in $A(v_P, l) \cap C(v_P, l)$. It follows that $|C(v_P, l)| \leq k + k^2$ a contradiction. □

5.4.2 FPT Algorithm for Trivial Dependency Scheme

We say that a dependency poset $\leq^V$ is *layered* if it there is a partition of the elements into *layers* $V_1, \dots, V_\ell$ such that for every $x_i \in V_i, x_j \in V_j$ we have $x_i < x_j$ if and only if $i < j$. Informally, such posets are divided into a chain of layers of incomparable elements. We note that every trivial dependency poset is a layered poset; however, other dependency posets may in some cases also be layered.

The remainder of this section is devoted to a proof of the following theorem.

Theorem 5.37. *Let $\leq^V$ be a layered dependency poset and k a natural number. There exists an FPT algorithm (parameterized by k) that either finds a prefix path decomposition of G w.r.t. $\leq^V$ of width at most k or determines that no such path decomposition exists.*

Our general proof strategy will be to first show that any prefix path decomposition w.r.t. a layered dependency poset has a simple structure, which then can be exploited for an FPT algorithm that finds such a decomposition of small width. In particular, we will show that any prefix path decomposition can be naturally divided into path decompositions of the parts $(V_1, \dots, V_l)$ given by a layered dependency poset that are held together via (minimal) separators between these parts.

In the following let G be a graph, $\leq^V$ be a layered dependency poset, let $(V_1, \dots, V_l)$ be the ordered partition associated with $\leq^V$, and let $\mathcal{P} = (P_1, \dots, P_m)$ be a prefix path decomposition of G w.r.t. $\leq^V$. Moreover, for every layer i with $1 \leq i \leq l$, we denote by $V_{\leq i}$ the set $\bigcup_{1 \leq j \leq i} V_j$ and similarly by $V_{>i}$ the set $V_{>i} = \bigcup_{i<j\leq l} V_j$. We will be interested in (minimal) separators between $V_{\leq i}$ and $V_{>i}$ in G. In particular, we will show that any such separator corresponds to a vertex cover in the bipartite graph $B(i)$ defined as follows.

- $V(B(i))$ contains the vertices of G that either lie in $V_{\leq i}$ and have at least one neighbor in $V_{>i}$, or are in $V_{>i}$ and have at least one neighbor in $V_{\leq i}$.
- $\{a, b\} \in E(B(i))$ if and only if $\{a, b\} \in E(G)$ and $a \in V_{\leq i}$ and $b \in V_{>i}$.

Proposition 5.38. *For every i with $1 \leq i \leq l$, it holds that a set C of vertices of G is a separator between $V_{\leq i}$ and $V_{>i}$ in G if and only if C is a vertex cover of $B(i)$.*

Proof. Towards showing the forward direction, let C be a separator between $V_{\leq i}$ and $V_{>i}$ in G. Then C is also a vertex cover of $B(i)$ since otherwise there would be an edge e between $V_{\leq i}$ and $V_{>i}$ in G with $C \cap e = \emptyset$, contradicting our assumption that C is a separator .

Towards showing the reverse direction, let C be a vertex cover of $B(i)$. Then C is also a separator between $V_{\leq i}$ and $V_{>i}$ in G since otherwise there would be an edge e between

$V_{\leq i}$ and $V_{>i}$ in $B(i)$ with $C \cap e = \emptyset$, contradicting our assumption that C is a vertex cover of $B(i)$. □

In the following we will show that for every i with $1 \leq i \leq l$, $\mathcal{P}$ contains at least one bag that contains all vertices of a minimal separator between $V_{\leq i}$ and $V_{>i}$ in G. Namely, we will show that this holds for the first bag of $\mathcal{P}$ which precedes a bag that forgets a vertex from $V_{>i}$. We will denote this bag by $p(i)$; formally, we set $p(i) = \min\{\, f_{\mathcal{P}}(v) \mid v \in V_{>i} \,\}$ where "min" returns the left-most bag out of the set.

Lemma 5.39. *For every i with $1 \leq i \leq l$, it holds that the bag $p(i)$ of $\mathcal{P}$ contains a minimal separator between $V_{\leq i} = \bigcup_{1 \leq j \leq i} V_j$ and $V_{>i} = \bigcup_{i<j\leq l} V_j$ in G.*

Proof. Because of Proposition 5.38 it is sufficient to show that the bag $p(i)$ contains a vertex cover of the bipartite graph $B(i)$. Towards showing this consider an edge $\{a, b\} \in E(B(i))$ with $a \in V_{\leq i}$ and $b \in V_{>i}$. It follows from the choice of $p(i)$ that:

O1 Every vertex in $V(B(i)) \cap V_{>i}$ that has been introduced in some bag of $\mathcal{P}$ to the left of $p(i)$ or in $p(i)$ is also in the bag $p(i)$,

O2 Every vertex in $V(B(i)) \cap V_{\leq i}$ has been introduced in some bag of $\mathcal{P}$ to the left of $p(i)$ or in $p(i)$.

If b is contained in the bag $p(i)$ then there is nothing to show. So suppose that b is not in the bag $p(i)$. It follows from Observation O1 that b has not yet been introduced. Hence, because of Property (T3) of a path decomposition, we obtain that the vertex a needs to occur in some bag to the right of $p(i)$ and it now follows from Observation O2 together with Property (T2) of a path decomposition that a is contained in the bag $p(i)$. □

Using Lemma 5.39, we are ready to show that there always exists an optimal prefix path decomposition of a graph G w.r.t. a dependency poset that is layered, which contains a minimal separator between $V_{\leq i}$ and $V_{>i}$ (for each $i \leq l$) in G as a bag.

Lemma 5.40. *There exists an optimal prefix path decomposition $\mathcal{Q}$ of G w.r.t. $\leq^V$, such that $\mathcal{Q} = (P_1, \ldots, P_{n_1}, \ldots, P_{n_2}, \ldots, P_{n_i}, \ldots, P_{n_l})$, where for every i:*

Q1 P_{n_i} is a minimal separator between $V_{\leq i}$ and $V_{>i}$;

Q2 $(P_{n_{i-1}}, \ldots, P_{n_i})$ is a path decomposition of $V_i \cup P_{n_{i-1}} \cup P_{n_i}$;

Q3 every vertex in $V_{>i}$ is forgotten after the bag P_{n_i}, and $V_{\leq i} \subseteq P_{\leq n_i}$.

Proof. We start with a given optimal prefix path decomposition $\mathcal{P}_0 = \mathcal{P}$ and we transform it inductively to prefix path decompositions $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_l$ such that $\mathcal{P}_j$ satisfies properties Q1, Q2 and Q3 up to index j. Clearly, $\mathcal{P}_0$ satisfies the conditions Q1, Q2 and Q3 up to index 0. As our inductive hypothesis, we assume we are given a prefix path decomposition $\mathcal{P}_{i-1} = (P_1, \ldots, P_{n_1}, \ldots, P_{n_2}, \ldots, P_{n_{i-1}}, \ldots P_r)$ which satisfies the conditions Q1, Q2 and Q3 up to $i-1$.

We proceed by showing how to construct $\mathcal{P}_i$ from $\mathcal{P}_{i-1}$. Let $P_{i'}$ be the bag $p(i)$ (i.e., the first bag of $\mathcal{P}_{i-1}$ that precedes the bag forgetting a vertex from $V_{>i}$). Since Q3 holds for $i-1$, it is easy to see that $i' \geq n_{i-1}$.

By Lemma 5.39 $P_{i'}$ contains a minimal separator X_i between $V_{\leq i}$ and $V_{>i}$. We set $X_{\leq i} = P_{i'} \cap (V_{\leq i} \setminus X_i)$, and $X_{>i} = P_{i'} \cap (V_{>i} \setminus X_i)$. Next, we construct $\mathcal{P}_i = (P'_1, \ldots, P'_{r+2})$ such that

- $P'_j = P_j$ for $j \leq n_{i-1}$,
- $P'_j = P_j \setminus X_{>i}$ for $n_{i-1} < j \leq i'$,
- $P'_{i'+1} = X_i$, and
- $P'_{j+2} = P_j \setminus X_{\leq i}$ for $j \geq i'$.

Moreover, we set $n_i = i' + 1$, hence Q1 holds for i. Since we do not change the path decomposition up to $P_{n_{i-1}}$, it is easy to see that Q1, Q2, Q3 still hold for all $j < i$. From the choice of $P_{i'}$ it follows that all vertices of $V_{>i}$ are forgotten after the bag $P'_{i'+1}$ and all vertices of $V_{\leq i}$ are introduced before the bag $P'_{i'+1}$ (because of the downward closure property). Hence, Q3 holds for i as well. Now we show that Q2 holds. Since Q3 holds for both $i-1$ and i, it follows that $P'_{n_{i-1}} \cup P'_{n_{i-1}+1} \cup \cdots \cup P'_{n_i} = V_i \cup P'_{n_{i-1}} \cup P'_{n_i}$. From the construction of $\mathcal{P}_i$, it follows that for every vertex x in $V_i \cup P'_{n_{i-1}} \cup P'_{n_i}$ it holds that if $x \in P_j$ for $n_{i-1} \leq j \leq i'$, then $x \in P'_j$. Hence, the bags containing x form an interval. We are left to show that every edge in $V_i \cup P'_{n_{i-1}} \cup P'_{n_i}$ is contained in some bag P_j for $n_{i-1} \leq j \leq i'$, hence it is also in the bag P'_j. Let e be an arbitrary edge with both endpoints in $V_i \cup P'_{n_{i-1}} \cup P'_{n_i}$. Since $\mathcal{P}_{i-1}$ is a path decomposition, there is a bag, say P_{j_e}, that contain both endpoints of e. However note that all vertices of $V_i \cup P'_{n_{i-1}} \cup P'_{n_i}$ are forgotten in $\mathcal{P}_{i-1}$ after the bag $P_{n_{i-1}}$ and introduced before the bag $P_{i'}$. Hence if $j_e < n_{i-1}$, then e is contained also in $P_{n_{i-1}}$ and similarly if $j_e > i'$ then e is also in the bag $P_{i'}$. Therefore, $\mathcal{P}_i$ satisfies all the conditions Q1, Q2, Q3 for i.

We are left to verify that $\mathcal{P}_i$ is also a prefix path decomposition w.r.t. $\leq^V$. It is easy to verify that since each vertex of G is in a bag of $\mathcal{P}_{i-1}$, the same holds for $\mathcal{P}_i$ as well.

Next, we show that property (T2) holds. The only way our operations could have violated monotonicity is by deletion of $X_{>i}$ from a certain interval of the prefix path decomposition, and in particular only for a vertex $x \in X_{>i}$ which occurs in $P_{n_{i-1}}$ and also in $P_{i'}$. By minimality of the separator in $P_{n_{i-1}}$ and Q3, there must exist a neighbor y of x in $V_{\leq i-1}$ which has already been forgotten in $P_{n_{i-1}}$. But then every minimal separator between $V_{\leq i}$ and $V_{>i}$ that is contained in $P_{i'}$ has to also contain x, since it has to cover the edge xy. As a consequence, $x \in X_i$ which contradicts $x \in X_{>i}$.

We proceed by showing that property (T3) holds. Since X_i is a minimal separator between $V_{>i}$ and $V_{\leq i}$, all edges between $X_{\leq i}$ and $V_{>i}$ are contained in the edges between $X_{\leq i}$ and X_i, and hence these are in the bag $P'_{i'}$. Similarly, all edges between $X_{>i}$ and $V_{\leq i}$ are in $P'_{i'+2}$. Furthermore, all vertices of $V_{>i}$ are forgotten after $P'_{i'+2}$ and all vertices of $V_{\leq i}$ are introduced before $P'_{i'}$. Therefore, the fact that every edge of G is in a bag of $\mathcal{P}_i$ follows from the construction of $\mathcal{P}_i$ and the fact that $\mathcal{P}_{i-1}$ is a path decomposition.

Finally, we show that $\mathcal{P}_i$ has the downward closure property. We consider three possibilities. If $v \in V_{\leq i} \setminus P_{i'}$, then clearly $f_{\mathcal{P}_i}(v) = P'_j$ for some $j < i'$ and $P'_{\leq j} = P_{\leq j}$ or $P'_{\leq j} = P_{\leq j} \setminus X_{>i}$. Since $v \in V_{\leq i}$ and all vertices in $X_{>i}$ are from $V_{>i}$, $D_{\leq}(v) \subseteq P_{\leq j}$ implies $D_{\leq}(v) \subseteq P'_{\leq j}$. If $v \in X_{\leq i}$, then $f_{\mathcal{P}_i}(v) = P'_{i'}$ and $D_{\leq}(v) \subseteq V_{\leq i} \subseteq P'_{\leq i'}$ follows from Q3. Otherwise, if $f_{\mathcal{P}_i}(v) = P'_j$, then $f_{\mathcal{P}_{i-1}}(v) = P_{j-2}$ and $D_{\leq}(v) \subseteq P'_{\leq j} = P_{\leq j-2}$. □

To later be able to compute a prefix path decomposition, we need to be able to enumerate all possible minimal separators between $V_{\leq i}$ and $V_{>i}$ for any i with $1 \leq i \leq l$. The next lemma shows that this is indeed possible.

Lemma 5.41. *For every i with $1 \leq i \leq l$, there are at most 2^k minimal separators between $V_{\leq i}$ and $V_{>i}$ in G and these can be enumerated in time $\mathcal{O}(|E(B(i))| + k^2 2^k)$.*

Proof. Follows immediately by combination of Proposition 5.38 and Lemma 5.33. □

Our proof requires one last technical observation, which states that it is possible to find path decompositions starting and ending with a specific bag.

Observation 5.42. *Given a graph G, integer k, and vertex subsets P, P' of size at most $k+1$, there exists an FPT algorithm (parameterized by k) which either constructs a path decompositions which starts with a bag P and ends with a bag P' of width at most k or determines that no such path decompositions exists.*

Proof. Let us represent an instance (G, P, P') as a labeled graph G' obtained from G by the addition of two labels: the label P denotes vertices in P and the label P' denotes vertices in P'. We will show that the existence of a path decomposition $\mathcal{P}$ with the required properties is closed under the minor operation on the labeled graph G'. Indeed, any path decomposition of G' can be modified to a path decomposition for any minor of G' in exactly the same way as in the case of standard path decompositions. Hence, it is sufficient to check (G, P, P') against some $f(k)$, for some function f, many forbidden labeled minors and the claim then follows from Corollary 7.2 of Kawarabayashi et al. [134]. □

We remark that a more efficient algorithm for Observation 5.42 could be obtained by adapting Bodlaender's algorithm [22]; however, a formal proof of this claim is outside the scope of this thesis.

Proof of Theorem 5.37. By Lemma 5.40 we can just concentrate on finding a prefix path decomposition such that $\mathcal{P} = (\emptyset = P_0, \dots, P_{n_1}, \dots, P_{n_2}, \dots, P_{n_i}, \dots, P_{n_l} = \emptyset)$, where P_{n_i} is a minimal separator between $V_{\leq i}$ and $V_{>i}$ and $(P_{n_{i-1}}, \dots, P_{n_i})$ is a prefix path decomposition of $V_i \cup P_{n_{i-1}} \cup P_{n_i}$. We say that minimal separators C between $V_{\leq i-1}$ and $V_{>i-1}$ and D between $V_{\leq i}$ and $V_{>i}$ are compatible if

- if C contains a vertex $v \in V_{>i}$, then D contains v.
- if D contains a vertex $v \in V_{<i}$, then C contains v.

Since by Lemma 5.40 (property Q3) all vertices in $V_{>i}$ are forgotten in $\mathcal{P}$ after the bag P_{n_i} and $V_{\leq i} \subseteq P_{\leq n_i}$ it follows that $P_{n_{i-1}}$ and P_{n_i} are compatible for all i. Since all vertices of V_i are incomparable, every proper path decomposition of $V_i \cup P_{n_{i-1}} \cup P_{n_i}$ starting with the bag $P_{n_{i-1}}$ and ending with the bag $P_{n_{i-1}}$ is also a prefix path decomposition. To decide whether G has such prefix path decomposition w.r.t. V of width k we first construct an auxiliary directed graph H as follows. H contains a vertex $(i, P, P', \mathcal{P})$ for every natural number i with $1 \leq i \leq l$ and for every two compatible minimal separators P, P' of size at most k between $V_{\leq i-1}$ and $V_{>i-1}$ and between $V_{\leq i}$ and $V_{>i}$, respectively, and $\mathcal{P}$ is a path decomposition of $V_i \cup P \cup P'$ of width at most k that starts with P and ends with P'.

Moreover, there are two special vertices $(0, \emptyset, \emptyset, \emptyset)$ and $(l+1, \emptyset, \emptyset, \emptyset)$. There is an edge in H from every vertex $(i, P, P', \mathcal{P})$ to any vertex $(i+1, P', P'', \mathcal{P}')$, where $0 \leq i \leq l+1$. By Lemma 5.41 there are at most 2^k minimal separators between $V_{\leq i}$ and $V_{>i}$ in G and these can be enumerated in time $\mathcal{O}(k^2 2^k + |E(B(i))|)$. Therefore, H has $\mathcal{O}(l \cdot 4^k)$ vertices. Moreover, for each i, we can enumerate all pairs P and P' and check their compatibility in time $\mathcal{O}(4^k + |E(G)|)$. Furthermore, for each such pair P, P', we can compute an optimal path decomposition of of $V_i \cup P \cup P'$ starting with P and ending with P' by Observation 5.42. Once H is constructed, it is easily observed that each path from $(0, \emptyset, \emptyset, \emptyset)$ to $(l+1, \emptyset, \emptyset, \emptyset)$ gives us a prefix path decomposition of G w.r.t. $\leq^V$ of width at most k. On the other hand, if there exists a prefix path decomposition of G w.r.t. $\leq^V$, then by Lemma 5.40 there exists one of the form $\mathcal{P} = (\emptyset = P_0, \dots, P_{n_1}, \dots, P_{n_2}, \dots, P_{n_i}, \dots, P_{n_l} = \emptyset)$, where $P_{n_{i-1}}$ and P_{n_i} are compatible minimal separators for all i. But then H contains a path $(0, \emptyset, \emptyset, \emptyset)$, $(0, \emptyset, P_{n_1}, \mathcal{P}_1)$, $(0, P_{n_1}, P_{n_2}, \mathcal{P}_2)$,..., $(l+1, P_{n_l}, \emptyset, \emptyset)$. Finally, finding such a path in H takes time time at most $|V(H)|^2$. □

5.5 Summary and Open Questions

Our results push the frontiers of tractability for QBF to new natural classes of instances. A number of interesting research questions still remain open in the area, and perhaps the most prominent of these is whether one can lift our results towards prefix treewidth. This would be especially interesting for posets of unbounded width, since on bounded-width posets these parameters differ only by a constant factor. The exact complexity of computing prefix treewidth and prefix pathwidth on general posets remains a challenging open problem.

Notes

The results in this chapter appeared in a conference paper in the proceedings of The Thirtieth AAAI Conference on Artificial Intelligence (AAAI 2016) [72].

Part II

Distance to Triviality

CHAPTER 6

Background

Over the past years, the parameterized complexity of finding a modulator to a plethora of graph classes has been systematically researched. In particular, the most attention has been devoted to the modulators to classes that admit efficient algorithms for many NP-hard problems with a hope to lift the algorithmic properties of these classes to obtain FPT algorithms [88, 92]. Therefore, problems such as Vertex Cover or Feedback Vertex Set are of immediate interest in this setting as modulators to edgeless graph and forests, respectively. These classes can be viewed as classes of graphs having treewidth 0 and 1, respectively. Furthermore, many problems are known to be solvable in polynomial time on graphs of a constant treewidth [46]. For this reason, classes of graphs of constant treewidth have been studied in detail, and Fomin et al. [88] and Kim et al. [138] showed that the corresponding Treewidth-t Vertex Deletion[1] problem is solvable in single-exponential FPT time and it even admits a linear kernel. This leads us to the first technical chapter of Part II, where we achieve the first step towards extending these results to modulators to graphs of constant rank-width. Namely, we design a single exponential FPT algorithm for Rank-width-1 Vertex Deletion.

Afterwards, we switch our attention towards directed graphs. Here the situation is more complicated. Even though there exists an equivalent notion of directed treewidth for directed graphs [19, 130, 135], an FPT algorithm for Directed Feedback Vertex Set, which is the problem of finding a modulator to directed acyclic graph (graphs of directed treewidth 0), was a longstanding open problem in the area of parameterized algorithms. Following the resolution of the fixed-parameter tractability in 2008 by Chen et al. [42], two prominent questions were raised: can the problem be solved in single-exponential FPT time, and does a polynomial kernel exist for the problem? These problems have since become two of the main open problems in the area of parameterized complexity. We address the latter question in Chapter 8. However, we were not able to answer the

[1] Treewidth-t Vertex Deletion asks whether it is possible to delete k vertices so that the resulting graph has treewidth at most t.

question in its full generality; instead, we give a polynomial kernel for a parameter that is always at least the size of directed feedback vertex set. Namely, we parameterized by the size of the feedback vertex set of the underlying undirected graph.

Finally, in the last chapter of Part II, we study an application of modulator-based techniques towards obtaining FPT algorithms for ILP. More precisely, we investigate what happens if the incidence graph of a given ILP instance has a small modulator to small components. Note that this parameter is another natural generalization of vertex cover, since vertex cover is precisely the modulator to components of size 1.

CHAPTER 7

Modulators to Distance-Hereditary Graphs

The successful development of single-exponential FPT algorithms for Treewidth-t Vertex Deletion motivates us to study Rank-width-t Vertex Deletion, which is analogous to Treewidth-t Vertex Deletion but replaces treewidth with the structural parameter rank-width. Kanté et al. [133] observed that Rank-width-t Vertex Deletion is fixed-parameter tractable using the general framework of Courcelle, Makowsky, and Rotics [50]. However, this algorithm does not provide any reasonable function for k. Thus Kanté et al. naturally asked whether it is solvable in reasonably better running time. For instance, it is actually open whether Rank-width-t Vertex Deletion can even be solved in time $2^{2^{\mathcal{O}(k)}}n^{\mathcal{O}(1)}$, where k is the size of the deletion set.

In this chapter, we focus on graphs of rank-width at most 1, which are precisely *distance-hereditary graphs* [167]. Bandelt and Mulder [17] found all the minimal induced subgraph obstructions for distance-hereditary graphs. Distance-hereditary graphs are naturally related to split decompositions, where they are exactly the graphs that are completely decomposable into stars and complete graphs [31].

Given the above, we view the vertex deletion problem for distance-hereditary graphs as a first step towards handling Rank-width-t Vertex Deletion.

Results

Recall that a graph G is called distance-hereditary if for every connected induced subgraph H of G and every $v, w \in V(H)$, the distance between v and w in H is the same as the distance between v and w in G. We study the following parameterized problem.

Distance-Hereditary Vertex Deletion
Instance: A graph G and an integer k.
Parameter: k.
Task: Is there a vertex set $Q \subseteq V(G)$ with $|Q| \leq k$ such that $G - Q$ is distance-hereditary?

The main result of this chapter is a single-exponential FPT algorithm for Distance-Hereditary Vertex Deletion.

Theorem 7.1. Distance-Hereditary Vertex Deletion *can be solved in time* $\mathcal{O}(37^k \cdot |V(G)|^7(|V(G)| + |E(G)|))$.

We note that this solves an open problem of Kanté, Kim, Kwon, and Paul [133]. The core of our approach exploits two distinct characterizations of distance-hereditary graphs: one by forbidden induced subgraphs (obstructions), and the other by admitting a special kind of split decomposition [51].

The algorithm can be conceptually divided into three parts.

1. **Iterative Compression**. This technique allows us to reduce the problem to the easier Disjoint Distance-Hereditary Vertex Deletion, where we assume that the instance additionally contains a certain form of advice to aid us in our computation. Specifically, this advice is a vertex deletion set S to distance-hereditary graphs which is disjoint from and slightly larger than the desired solution.
2. **Branching Rules**. We exhaustively apply two branching rules to simplify the given instance of Disjoint Distance-Hereditary Vertex Deletion. At a high level, these branching rules allow us to assume that the resulting instance contains no small obstructions and furthermore that certain connectivity conditions hold on $G[S]$.
3. **Simplification of Split Decomposition**. We compute the split decomposition of $G - S$ and exploit the properties of our instance G guaranteed by branching to prune the decomposition. In particular, we show that the connectivity conditions and non-existence of small obstructions mean that S must interact with the split decomposition of $G - S$ in a special way, and this allows us to identify irrelevant vertices in $G - S$. This is by far the most technically challenging part of the algorithm.

A more detailed explanation of our algorithm is provided in Section 7.3, after the definition of required notions. We complement this result with an algorithmic lower bound which rules out a subexponential FPT algorithm for Distance-Hereditary Vertex Deletion under well-established complexity assumptions. We also note that the naive approach of simply hitting all known "obstructions" (i.e., forbidden induced subgraphs) for distance-hereditary graphs does not lead to an FPT algorithm. Indeed, the set of induced subgraph obstructions for distance-hereditary graphs includes induced cycles of length at least 5. Heggernes et al. [115] showed that the problem asking whether it is possible to delete k vertices so that the resulting graph has no induced cycles of length at least 5 is W[2]-hard. Therefore, unless W[2]=FPT, it is not straightforward to obtain a single-exponential FPT algorithm for Distance-Hereditary Vertex Deletion by simply finding and hitting all forbidden induced subgraphs for the class.

Organization of the Chapter

The Chapter is organized as follows. We begin with a short section introducing distance-hereditary graphs. Afterwards, in Section 7.2, we define splits and split decompositions. In Section 7.3, we set the stage for the process of simplifying the split decomposition, which entails the definition of Disjoint Distance-Hereditary Vertex Deletion, introduction of our branching rules, and a few technical lemmas which will be useful throughout the later sections. Section 7.4 then introduces and proves the soundness of five polynomial-time reduction rules; crucially, the exhaustive application of these rules guarantees that the resulting instance will have a certain “inseparability” property. Using this structural result, we prove that one of reduction rules is applicable until the remaining instance is trivial. Finally, the proof of our main result as well as the corresponding lower bound are presented in Section 7.5. Section 7.5 also illustrates that once we have an single-exponential FPT algorithm for finding the vertex deletion set to distance-hereditary graphs, it can be used as a parameter which allows single-exponential FPT algorithms for classical NP-hard problems.

7.1 Distance-Hereditary Graphs

A graph G is called distance-hereditary if for every connected induced subgraph H of G and every $v, w \in V(H)$, the distance between v and w in H is the same as the distance between v and w in G. For instance, the induced cycle $c_1c_2c_3c_4c_5c_1$ is not distance-hereditary, because the distance from c_1 to c_3 is 2, but if we take an induced subgraph on $\{c_1, c_3, c_4, c_5\}$, then the distance becomes 3. This graph class was first introduced by Howorka [119], and deeply studied by Bandelt and Mulder [17]. There are several other, equivalent characterizations of distance-hereditary graphs. Here we will exploit two other characterizations of the graph class: one by forbidden induced subgraphs (given below), and one via split decompositions (given in the following subsection).

The house, the gem, and the domino graphs are depicted in Figure 7.1. A graph isomorphic to one of the house, the gem, the domino, and induced cycles of length at least 5 will be called a *distance-hereditary obstruction* or shortly a *DH obstruction*. A DH obstruction with at most 6 vertices will be called a small DH obstruction. Two vertices v and w in a graph G are called twins if they have the same set of neighbors in $V(G) \setminus \{v, w\}$. Note that every DH obstruction does not contain any twins.

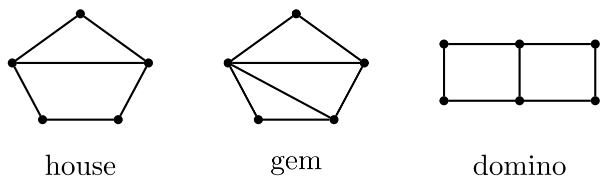


Figure 7.1: Small DH obstructions which are not cycles.

Theorem 7.2 (Bandelt and Mulder [17])**.** *A graph is distance-hereditary if and only if it contains no DH obstructions as induced subgraphs.*

We state an observation which will be useful later on.

Observation 7.3. *For any DH obstruction H and any edge e in H, it holds that the graph H' obtained by subdividing e also contains a DH obstruction as an induced subgraph.*

The following lemma will be used to find DH obstructions later on.

Lemma 7.4 (Kantè, Kim, Kwon, and Paul, Lemma 4.3 of [133])**.** *Let G be a graph obtained from an induced path of length at least 3 by adding a vertex v adjacent to its end vertices where v may be adjacent to some internal vertices of the path. Then G has a DH obstruction containing v. In particular, if the given path has length at most 4, then G has a small DH obstruction containing v.*

Proof. The first statement was shown in Lemma 4.3 of [133]. If the given path has length at most 4, then G has at most 6 vertices, and thus G contains a small DH obstruction containing v. □

7.2 Split decompositions

We follow the notations used by Bouchet [31].

For two vertex sets A and B, we say that

- A is complete to B if for every $a \in A$, $b \in B$, a is adjacent to b,
- A is anti-complete to B if for every $a \in A$, $b \in B$, a is not adjacent to b.

A split of a connected graph G is a vertex partition (X, Y) of G such that $|X| \geq 2, |Y| \geq 2$, and $N_G(Y)$ is complete to $N_G(X)$. See Figure 7.2 for an example. Splits are also called *1-joins*, or simply *joins* [91]. A connected graph G is called a prime graph if $|V(G)| \geq 5$ and it has no split. A bipartition is *trivial* if one of its parts is the empty set or a singleton. Cliques and stars are called degenerate graphs; notice that every non-trivial bipartition of their vertices is a split.

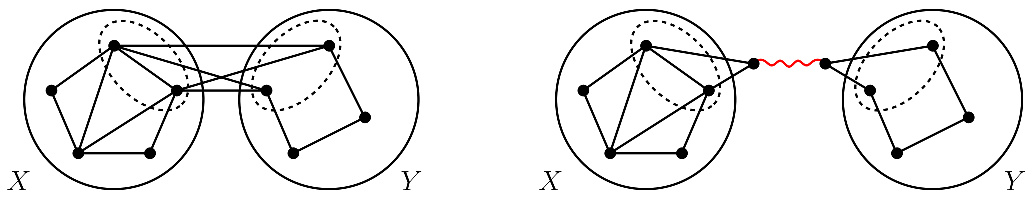


Figure 7.2: An example of a split (X, Y) of a graph. Its simple decomposition is presented in the second picture, where the red edge is the newly introduced marked edge.

A connected graph D with a distinguished set of edges $M(D)$ is called a marked graph if the edges in $M(D)$ form a matching and each edge in $M(D)$ is a cut edge. An edge in $M(D)$ is called a marked edge, and every other edge is called an unmarked edge. A vertex incident with a marked edge is called a marked vertex, and every other vertex is called an unmarked vertex. Each connected component of $D - M(D)$ is called a *bag* of D.

When a connected marked graph G, which will be a bag of a marked graph, admits a split (X, Y), we construct a marked graph D on the vertex set $V(G) \cup \{x', y'\}$ such that

- for vertices x, y with $\{x, y\} \subseteq X$ or $\{x, y\} \subseteq Y$, $xy \in E(G)$ if and only if $xy \in E(D)$,
- $x'y'$ is a new marked edge,
- X is anti-complete to Y,
- $\{x'\}$ is complete to $N_G(Y)$ and $\{y'\}$ is complete to $N_G(X)$ (with unmarked edges).

The marked graph D is called a *simple decomposition of* G. A split decomposition of a connected graph G is a marked graph D defined inductively to be either G or a marked graph defined from a split decomposition D' of G by replacing a connected component H of $D' - M(D')$ with a simple decomposition of H. See Figure 7.3 for an example of a split decomposition. The following lemma provides an important property. An example of an alternating path described in Lemma 7.5 is presented in Figure 7.3.

Lemma 7.5 (See Lemma 2.10 of Adler, Kanté, and Kwon [3])**.** *Let D be a split decomposition of a connected graph G and u, v be two vertices in G. Then $uv \in E(G)$ if and only if there is a path from u to v in D where its first and last edges are unmarked, and an unmarked edge and a marked edge alternatively appear in the path.*

Naturally, we can define a reverse operation of decomposing into a simple decomposition; for a marked edge xy of a split decomposition D, *recomposing* xy is the operation of removing two vertices x and y and making $N_D(x) \setminus \{y\}$ complete to $N_D(y) \setminus \{x\}$ with unmarked edges. It is not hard to observe that if D is a split decomposition of G, then G can be obtained from D by recomposing all marked edges.

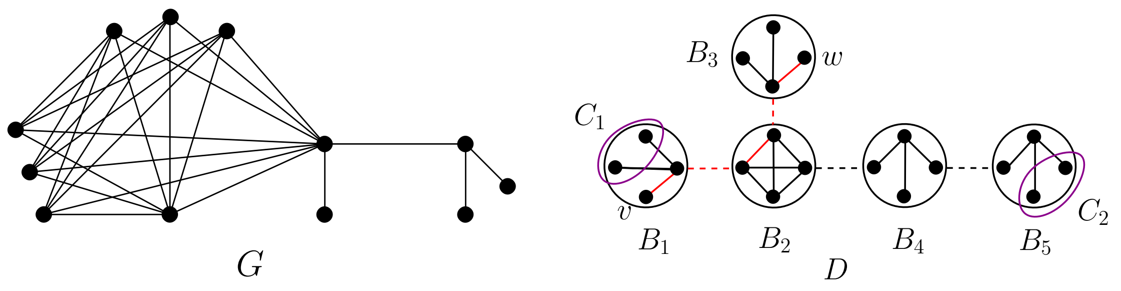


Figure 7.3: A graph G and its canonical split decomposition D. Marked edges are represented by dashed edges, and bags are indicated by circles. Note that path$(B_1, B_5) = \{B_1, B_2, B_4, B_5\}$ and $\{B_4, B_5\}$ is the set of (C_1, C_2)-separator bags, and $\{B_4\}$ is the set of (B_1, B_5)-separator bags. The shortest path from v to w in D is a path from v to w where its first and last edges are unmarked, and an unmarked edge and a marked edge alternatively appear in the path. The existence of such a path exactly corresponds to the adjacency relation in the original graph. The distance between C_1 and C_2 in G is 3, and there are two (C_1, C_2)-separator bags.

Note that there are many ways of decomposing a complete graph or a star, because every non-trivial vertex partition of these graphs is a split. Cunningham and Edmonds [52] developed a canonical way to decompose a graph into a split decomposition by not allowing to decompose a bag which is a star or a complete graph. A split decomposition D of G is called a canonical split decomposition if each bag of D is either a prime graph, a star, or a complete graph, and every recomposing of a marked edge in D results in a split decomposition without the same property. It is not hard to observe that every canonical split decomposition has no marked edge linking two complete bags, and no marked edge linking a leaf of a star bag and the center of another star bag [31]. Furthermore, for each pair of twins a and b in G, it holds that a and b must both be located in the same bag of the canonical split decomposition.

Theorem 7.6 (Cunningham and Edmonds [52]). *Every connected graph has a unique canonical split decomposition, up to isomorphism.*

Theorem 7.7 (Dahlhaus [54]). *The canonical split decomposition of a graph G can be computed in time $\mathcal{O}(|V(G)| + |E(G)|)$.*

We can now give the second characterization of distance-hereditary graphs that is crucial for our results. For convenience, we call a bag a star bag or a complete bag if it is a star or a complete graph, respectively.

Theorem 7.8 (Bouchet [31]). *A graph is a distance-hereditary graph if and only if every bag in its canonical split decomposition is either a star bag or a complete bag.*

We will later on also need a little bit of additional notation related to split decompositions of distance-hereditary graphs. Let D be a canonical split decomposition of a distance-hereditary graph. For two distinct bags B_1 and B_2, we denote by $\mathrm{comp}(B_1, B_2)$ the connected component of $D - V(B_1)$ containing B_2. Technically, when $B_1 = B_2$, we define $\mathrm{comp}(B_1, B_2)$ to be the empty set. For two bags B_1 and B_2, we denote by $\mathrm{path}(B_1, B_2)$ the set of all bags containing a vertex in a shortest path from B_1 to B_2 in D. In other words, when we obtain a tree from D by contracting every bag B into a node $v(B)$, $\mathrm{path}(B_1, B_2)$ is the set of all bags corresponding to nodes of the unique path from $v(B_1)$ to $v(B_2)$ in the tree. See Figure 7.3.

Let C_1 and C_2 be two disjoint vertex subsets of D such that C_1 and C_2 are sets of unmarked vertices contained in (not necessarily distinct) bags B_1 and B_2, respectively. A bag B is called *a* (C_1, C_2)*-separator bag* if B is a star bag contained in $\mathrm{path}(B_1, B_2)$ whose center is adjacent[1] to neither $\mathrm{comp}(B, B_1)$ nor $\mathrm{comp}(B, B_2)$. We remark that B can be B_i for some $i \in \{1, 2\}$, and especially when $B = B_1 = B_2$, B is a star bag and each C_i consists of leaves of B and B_1 is the unique (C_1, C_2)-separator bag. For convenience, we also say that a bag B is *a* (B_1, B_2)*-separator bag* if B is a star bag contained in $\mathrm{path}(B_1, B_2) \setminus \{B_1, B_2\}$ whose center is adjacent to neither $\mathrm{comp}(B, B_1)$ nor $\mathrm{comp}(B, B_2)$. For this notation, B cannot be B_1 nor B_2.

Observation 7.9. *The distance between C_1 and C_2 in the original graph is exactly the same as one plus the number of* (C_1, C_2)*-separator bags.*

7.3 Setting the Stage

We begin by applying the iterative compression technique, first introduced by Reed, Smith and Vetta [174] to show that Odd Cycle Transversal can be solved in single-exponential FPT time. This technique allows us to transform our original target problem to one that is easier to handle, which we call Disjoint Distance-Hereditary Vertex Deletion. Our goal for the majority of the chapter will be to obtain a single-exponential FPT algorithm for Disjoint Distance-Hereditary Vertex Deletion; this is then used to obtain an algorithm for Distance-Hereditary Vertex Deletion in Section 7.5.

Disjoint Distance-Hereditary Vertex Deletion
Instance : A graph G, an integer k, and $S \subseteq V(G)$ such that $G - S$ is distance-hereditary.
Task : Is there $Q \subseteq V(G) \setminus S$ with $|Q| \leq k$ such that $G - Q$ is distance-hereditary?

We will denote an instance of Disjoint Distance-Hereditary Vertex Deletion as a tuple (G, S, k). The major part of our result is to prove that this problem can be solved in time $2^{\mathcal{O}(k + \mathrm{cc}(G[S]))} n^{\mathcal{O}(1)}$, where $\mathrm{cc}(G[S])$ denotes the number of connected components of $G[S]$. We note that any instance of Disjoint Distance-Hereditary Vertex Deletion such that $G[S]$ is not distance-hereditary must clearly be a No-instance; hence we will assume that we reject all such instances immediately.

[1]Recall that a vertex v is adjacent to $S \subseteq V(G)$, if it is adjacent to some vertex $u \in S$.

Before explaining the general approach for solving Disjoint Distance-Hereditary Vertex Deletion, it will be useful to introduce a few definitions. Since the canonical split decomposition guaranteed by Theorem 7.8 only helps us classify twins in $G - S$ and not in G, we explicitly define an equivalence $\sim$ on the vertices of $G - S$ which allows us to classify twins in G:

$$\text{for two vertices } u, v \in V(G - S),\ u \sim v \text{ iff they are twins in } G.$$

We denote by $\mathbf{tc}(G - S)$ the set of equivalence classes of $\sim$ on $V(G - S)$, and each individual equivalence class will be called a twin class in $G - S$. We can observe that if $U \in \mathbf{tc}(G - S)$ lies in a single connected component of $G - S$, then U must be contained in precisely one bag of the split decomposition of this connected component of $G - S$, as U is a set of twins in $G - S$ as well. A twin class is *S-attached* if it has a neighbor in S, and *non-S-attached* if it has no neighbors in S. Similarly, we say that a bag in the canonical split decomposition of $G - S$ is *S-attached* if it has a neighbor in S, and *non-S-attached* otherwise.

We frequently use a special type of star bags. A star bag B is called *simple* if its center is either unmarked or adjacent to a connected component of $D - V(B)$ consisting of one non-S-attached bag.

7.3.1 Overview of the Approach

Now that we have introduced the required terminology, we can provide a high-level overview of our approach for solving Disjoint Distance-Hereditary Vertex Deletion.

1. We exhaustively apply the branching rules described in Section 7.3.2. Branching rules will be applied when G has a small subset $X \subseteq V(G - S)$ such that $S \cup X$ induces a DH obstruction, or there is a small connected subset $X \subseteq V(G - S)$ such that adding X to S decreases the number of connected components in $G[S]$.

2. We exhaustively apply the initial reduction rules described in Section 7.4. Each of these rules runs in polynomial time, finds a part in the canonical split decomposition of a connected component of $G - S$ that can be simplified, and modifies the decomposition. Each application of a reduction rule from Section 7.4 either reduces the number of vertices in $G - S$ or reduces the total number of bags in the canonical split decomposition (of a connected component of $G - S$). It is well known that the total number of bags in the canonical split decomposition of a graph is linear in the number of vertices. Therefore, the total number of application of these initial reduction rules will also be at most linear in the number of vertices.

3. We say that G and the canonical split decompositions of $G - S$ are *reduced* if the branching rules in Section 7.3.2 and reduction rules in Section 7.4 cannot be applied

anymore. We will obtain the following simple structure of the decompositions in the reduced instance:

- Each canonical split decomposition D of a connected component of $G - S$ contains at least two distinct S-attached twin classes (Reduction Rule 7.1).
- Each bag contains at most one S-attached twin class (Lemma 7.18).
- When B is a bag and D' is a connected component of $D - V(B)$ containing no bags having a neighbor in S, D' consists of one bag and B is a star bag whose center is adjacent to D' (Lemma 7.19). In this case, B is a simple star bag whose center is adjacent to D'.
- When B is a bag and D' is a connected component of $D - V(B)$ such that D' contains exactly one S-attached bag B', there is no (B', B)-separator bag (Lemma 7.26).

4. Using these structures, we prove in Subsection 7.4.4 that if a split decomposition of a connected component of $G - S$ contains two S-attached twin classes, then one of reduction rules should be applied. For this, we assign any bag as a root bag R of D and choose a bag B with maximum $|\text{path}(B, R)|$ such that there are two descendant bags of B having S-attached twin classes C_1 and C_2, respectively. Then the distance from C_1 to C_2 in $G - S$ is at most 2, and thus their neighbors on S should be close to each other, as branching rules cannot be applied further. Depending on the type of B and the distance from C_1 to C_2, we show separately that one of reduction rules can be applied.

   It will imply that we can apply one of all rules recursively until $G - S$ is empty or k becomes 0. Then we can test whether the resulting instance is distance-hereditary or not in polynomial time, and output an answer.

Let us also say a few words about the running time of the algorithm. Let $\mu := k + \text{cc}(G[S])$. Each of our branching rules will reduce μ and branch into at most 6 subinstances. Each reduction rule takes polynomial time, and the reduction rules will be applied at most $\mathcal{O}(|V(G)|)$ times. Whenever we introduce a new rule, we also show that it is *sound*.

A vertex v in $G - S$ is called *irrelevant* if (G, S, k) is a YES-instance if and only if $(G - v, S, k)$ is a YES-instance. We will be identifying and removing irrelevant vertices in several of our reduction rules. When removing a vertex v from $G - S$, it is easy to modify the canonical split decomposition containing v, and thus it is not necessary to recompute the canonical split decomposition of the resulting graph from scratch. More details regarding such modifications of split decompositions can be found in the work of Gioan and Paul [105].

7.3.2 Branching Rules

We state our two branching rules below.

Branching Rule 7.1. *For every vertex subset X of $G - S$ with $|X| \leq 5$, if $G[S \cup X]$ is not distance-hereditary, then we remove one of the vertices in X, and reduce k by 1.*

Branching Rule 7.2. *For every vertex subset X of $G - S$ with $|X| \leq 5$ such that $G[X]$ is connected and the set $N_G(X) \cap S$ is not contained in a connected component of $G[S]$, then we either remove one of the vertices in X and reduce k by 1, or put all of them into S (which reduces the number of connected components of $G[S]$).*

The soundness of Branching Rules 7.1 and 7.2 are clear, and these rules can be performed in polynomial time. The exhaustive application of these branching rules guarantees the following structure of the instance.

Lemma 7.10. *Let (G, S, k) be an instance reduced under Branching Rules 7.1 and 7.2.*

(1) G has no small DH obstructions.

(2) Let $v \in V(G - S)$. For every two vertices $x, y \in N_G(v) \cap S$, they are contained in the same connected component of $G[S]$ and there is no induced path of length at least 3 from x to y in $G[S]$. Specifically, if $xy \notin E(G)$, then there is an induced path xpy for some $p \in S$.

(3) There is no induced path $v_1 \cdots v_5$ of length 4 in $G - S$ where v_1 and v_5 have neighbors in S but v_2 and v_4 have no neighbors in S.

(4) There is no induced path $v_1 \cdots v_4$ of length 3 in $G - S$ where v_1 and v_4 have neighbors in S but v_2 has no neighbors in S.

Proof. (1) Suppose G has a small DH obstruction H. Since $G - S$ is distance-hereditary, $V(H) \cap S \neq \emptyset$. Thus, $|V(H) \setminus S| \leq 5$, and it can be reduced under Branching Rule 7.1.

(2) First, by Branching Rule 7.2, x and y are contained in the same connected component of $G[S]$. Suppose there is an induced path of length at least 3 from x to y in $G[S]$. Then by Lemma 7.4, $G[S \cup \{v\}]$ contains a DH obstruction, contradicting our assumption that G is reduced under Branching Rule 7.1. So, if $xy \notin E(G)$, then there is an induced path of length 2 from x to y in $G[S]$.

(3) Suppose there is an induced path $v_1 \cdots v_5$ of length 4 in $G - S$ where v_1 and v_5 have neighbors on S but v_2 and v_4 have no neighbors on S. By Branching Rule 7.2, we know that $N_G(v_1) \cap S$ and $N_G(v_5) \cap S$ are contained in the same connected component of $G[S]$. Let P be a shortest path from $N_G(v_1) \cap S$ to $N_G(v_5) \cap S$ (if v_1 and v_5 have a common neighbor, then we choose a common neighbor). Then $v_2 v_1 P v_5 v_4$ is an induced path of length at least 4 and v_3 is adjacent to its end vertices. So, $G[S \cup \{v_1, \ldots, v_5\}]$ contains a DH obstruction, contradicting our assumption that G is reduced under Branching Rule 7.1.

(4) The same argument in (3) holds. □

Lemma 7.10, and especially point (2) in the lemma, is used in many parts of our proofs. Since we will apply the branching rules exhaustively at the beginning and also after each new application of a reduction rule, these properties will be implicitly assumed to hold in subsequent sections.

We will make use of two more lemmas based on our branching rules. These will be used in Section 7.4.4 as well as in the proof of Lemma 7.18 in Section 7.4.3.

Lemma 7.11. *Let (G, S, k) be an instance reduced under Branching Rules 7.1 and 7.2. Let C_1, C_2 be two distinct S-attached twin classes of $G - S$ such that C_1 is anti-complete to C_2, and $(N_G(C_1) \cap N_G(C_2)) \cap V(G - S) \neq \emptyset$. Then:*

(1) $(N_G(C_1) \cap N_G(C_2)) \cap S \neq \emptyset$.

(2) For every $x \in N_G(C_1) \setminus N_G(C_2)$ and every $y_1, y_2 \in N_G(C_1) \cap N_G(C_2)$, if x is adjacent to y_1, then x is adjacent to y_2 as well. It implies that x is adjacent to either all of vertices in $N_G(C_1) \cap N_G(C_2)$ or neither of them.

(3) For every $x \in N_G(C_1) \setminus N_G(C_2)$ and every $y_1, y_2 \in N_G(C_1) \cap N_G(C_2)$, if there is a path xpy_1 for some $p \in S \setminus N_G(C_1)$ (not necessarily induced), then p is adjacent to y_2 as well. It implies that p is adjacent to either all of vertices in $N_G(C_1) \cap N_G(C_2)$ or neither of them.

Proof. For each $i \in \{1, 2\}$ let $a_i \in C_i$ and let $T_i = N_G(C_i)$.

(1) Suppose $T_1 \cap S$ and $T_2 \cap S$ are disjoint. Let us choose a vertex z in $(T_1 \cap T_2) \cap V(G - S)$, which is not an empty set by assumption. Thus, $\{a_1, a_2, z\}$ induces a connected subgraph of G. If $T_1 \cap S$ and $T_2 \cap S$ are not contained in one connected component of $G[S]$, then we can apply Branching Rule 7.2. As our instance was reduced under Branching Rule 7.2, we know that $T_1 \cap S$ and $T_2 \cap S$ are contained in the same connected component of $G[S]$.

Let P be a shortest path from $T_1 \cap S$ to $T_2 \cap S$ in $G[S]$. Clearly, P contains at most one vertex from each $T_i \cap S$. As C_1 is anti-complete to C_2, $a_1 P a_2$ is an induced path of length at least 3 and z is adjacent to its end vertices. By Lemma 7.4, $G[V(P) \cup \{a_1, a_2, z\}]$ contains a DH obstruction, contradicting the assumption that G is reduced under Branching Rule 7.1.

(2) For contradiction, suppose $xy_1 \in E(G)$ and $xy_2 \notin E(G)$. Then $xa_1y_2a_2$ is an induced path of length 3 and y_1 is adjacent to its end vertices. By Lemma 7.4, G contains a small DH obstruction, contradiction.

(3) Suppose there is a path xpy_1 for some $p \in S \setminus T_1$ and p is not adjacent to y_2. First assume that $p \in S \setminus (T_1 \cup T_2)$. If $xy_2 \in E(G)$, then pxy_2a_2 is an induced path, and otherwise, $pxa_1y_2a_2$ is an induced path. Since y_1 is adjacent to p and a_2, by Lemma 7.4, G contains a small DH obstruction, contradiction. When $p \in (T_2 \setminus T_1) \cap S$, a_1xpa_2 becomes an induced path of length 3 and y_1 is adjacent to its end vertices, and thus G contains a small DH obstruction. We conclude that p is adjacent to y_2. □

Lemma 7.12. *Let (G, S, k) be an instance reduced under Branching Rules 7.1, and 7.2. Let C_1, C_2 be two distinct twin classes of $G - S$ such that C_1 is complete to C_2. Then:*

(1) For every $x \in N_G(C_1) \setminus (C_2 \cup N_G(C_2))$ and every $y_1, y_2 \in N_G(C_1) \cap N_G(C_2)$, if x is adjacent to y_1, then either it is adjacent to y_2 as well, or y_1 is adjacent to y_2.

(2) For every $x \in N_G(C_1) \setminus (C_2 \cup N_G(C_2))$ and every $y \in N_G(C_1) \cap N_G(C_2)$, if there is a path xpy for some $p \in V(G) \setminus (C_1 \cup N_G(C_1))$ (not necessarily induced), then $p \in N_G(C_2) \setminus N_G(C_1)$.

Proof. For each $i \in \{1, 2\}$ and let $a_i \in C_i$ and let $T_i = N_G(C_i)$.

(1) Suppose $xy_1 \in E(G)$ and $xy_2, y_1y_2 \notin E(G)$. Then $G[\{x, y_1, y_2, a_1, a_2\}]$ is isomorphic to the gem, contradiction.

(2) Suppose there is a path xpy for some $p \in V(G) \setminus (C_1 \cup N_G(C_1) \cup N_G(C_2))$. Then pxa_1a_2 is an induced path of length 3, and y is adjacent to its end vertices. By Lemma 7.4, $G[\{x, p, y, a_1, a_2\}]$ contains a small DH obstruction, contradiction. □

7.4 Reduction Rules in Split Decompositions

In this section, we assume that the given instance (G, S, k) is reduced under Branching Rules 7.1 and 7.2. The reduction rules introduced here either remove some irrelevant vertex, move some vertex into S, or reduce the number of bags in the decomposition by modifying the instance into an equivalent instance. After we apply any of these reduction rules, we will run the two branching rules from Section 7.3 again.

In Subsection 7.4.1, we introduce the notion of a *bypassing vertex*, which is a crucial concept that will frequently appear in our proofs. In Subsection 7.4.2, we present five reduction rules and prove their correctness. Then in Subsection 7.4.3, we discuss structural properties of the obtained instance after exhaustive application of all of the presented branching rules and reduction rules. These properties will be used in Section 7.4.4 to argue that if the instance is non-trivial, then one can apply one of reduction rules.

7.4.1 Bypassing Vertices

We introduce a generic way of finding an irrelevant vertex which will be used in many reduction rules. For a vertex v in $G - S$ and an induced path $H = p_1p_2p_3p_4p_5$ in G where $p_3 = v$, a vertex x in S is called a bypassing vertex for H and v if x is adjacent to p_2 and p_4 (see Figure 7.4). When H is clear from the context, we simply say that x is a bypassing vertex for v. If such a vertex x exists, it is clear that x is not contained in H. The following property is essential.

Lemma 7.13. *Let (G, S, k) be an instance reduced under Branching Rules 7.1 and 7.2. Let v be a vertex in $G - S$ such that for every induced path $P = p_1p_2p_3p_4p_5$ where $v = p_3$, there is a bypassing vertex for P and v. Then v is irrelevant.*

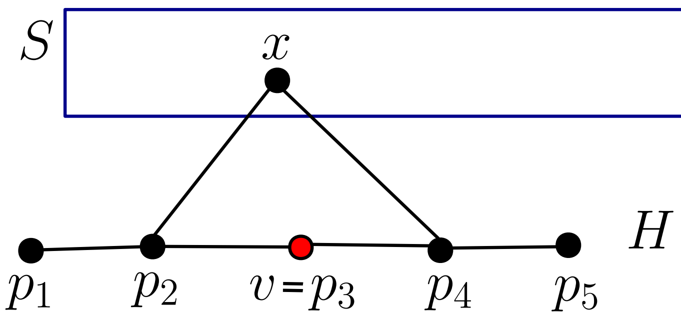


Figure 7.4: A bypassing vertex x for the induced path $H = p_1p_2p_3p_4p_5$ and the vertex v.

Proof. We claim that (G, S, k) is a YES-instance if and only if $(G - v, S, k)$ is a YES-instance. The forward direction is clear. Suppose that $G - v$ has a vertex set T such that $S \cap T \neq \emptyset$, $|T| \leq k$, and $(G - v) - T$ is distance-hereditary. If $G - T$ is distance-hereditary, then we are done. Suppose that $G - T$ has a DH obstruction H. Since Branching Rule 7.1 does not apply, G has no small DH obstructions, and therefore H is an induced cycle of length at least 7. Let $P = p_1p_2p_3p_4p_5$ be the subpath of H such that $p_3 = v$. By the assumption, there is a bypassing vertex v' for P and v. Note that $v' \notin V(H)$, as $v'p_2p_3p_4v'$ would be a cycle of length 4. Also, $H - v$ is an induced path of length at least 5. Thus $G[(V(H) \setminus \{v\}) \cup \{v'\}]$ contains another DH obstruction by Lemma 7.4. Note that $v' \in S$ and hence also $v' \in G - T$ and, in particular, $(V(H) \setminus \{v\}) \cup \{v'\} \subseteq V((G - v) - T)$. This contradicts the fact that $(G - v) - T$ is distance-hereditary. $\square$

Lemma 7.14. *Let (G, S, k) be an instance reduced under Branching Rules 7.1 and 7.2. Let v be a vertex in $G - S$ and $P = p_1p_2p_3p_4p_5$ be an induced path where $p_3 = v$. If $p_2, p_4 \in S$, then there is a bypassing vertex for P and v.*

Proof. Note that $p_2p_4 \notin E(G)$. Thus, by (2) of Lemma 7.10, there is an induced path p_2pp_4 for some $p \in S$, and p is a bypassing vertex. $\square$

7.4.2 Five Reduction Rules

We are now ready to start with our reduction rules. For the remainder of this section, let us fix a canonical split decomposition D of a connected component of $G - S$.

We start with a simple reduction rule that can be applied when D contains at most one S-attached twin class.

Reduction Rule 7.1. If D has at most one S-attached twin class, then we remove all unmarked vertices of D from G.

Proof of soundness. If D has no S-attached twin class, then its underlying graph is a distance-hereditary connected component of G. Thus, we can safely remove all its unmarked vertices. We may assume that D has one S-attached twin class C.

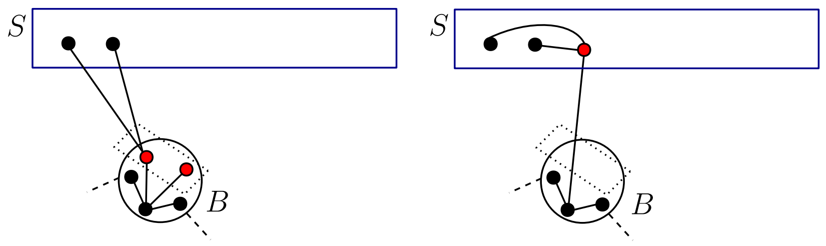


Figure 7.5: Reduction Rule 7.2.

Since C is the only S-attached twin class and every induced cycle of length at least 5 contains no twins, no induced cycle of length at least 5 contains an unmarked vertex in $V(D) \setminus C$. Thus, we can safely remove all of unmarked vertices other than vertices in C. Now we assume that $V(D) = C$. We claim that every vertex in C is also irrelevant.

To apply Lemma 7.13, suppose there is an induced path $P = p_1p_2p_3p_4p_5$ where $p_3 \in C$. Since there are no twins in P, P contains at most one vertex of C. Thus, p_2 and p_4 are contained in S. Since $p_2p_4 \notin E(G)$ and (G, S, k) is reduced under Branching Rules 7.1 and 7.2, by Lemma 7.14, there is a bypassing vertex for P and v. Since P was arbitrarily chosen, by Lemma 7.13, v is irrelevant. □

The next rule deals with a vertex of degree 1 in $G - S$. See Figure 7.5 for an illustration.

Reduction Rule 7.2. Let B be a star bag whose center is unmarked, and let v be an unmarked leaf in B. If v has no neighbor in S, then we remove v. If v has a neighbor in S, then we move v into S.

Proof of soundness. Let x be the center of B and let v be an unmarked leaf in B. If v has no neighbor in S, then v has degree 1 in G, and we can safely remove it. We assume that v has a neighbor in S. We claim that (G, S, k) is a Yes-instance if and only if $(G, S \cup \{v\}, k)$ is a Yes-instance. The converse direction is easy. Suppose that G contains a vertex set T where $S \cap T = \emptyset$, $|T| \le k$, and $G - T$ is distance-hereditary. Let $T' = T$ if $v \notin T$, and otherwise, we remove v from T and add x to T, and call it T'. We claim that $G - T'$ is distance-hereditary, which implies that $(G, S \cup \{v\}, k)$ is a Yes-instance.

Suppose $G - T'$ is not distance-hereditary. Since G has no small DH obstructions, $G - T'$ contains an induced cycle H of length at least 7. First assume that H contains x. Then x is not contained in T', and therefore, v was not contained in T, and we have $T = T'$ by the construction. Thus $G - T$ also contains H, contradiction. Thus, we have $x \notin V(H)$. If $v \notin V(H)$, then H is an induced subgraph of $G - T$ because T and T' only differ at $\{v, x\}$. This implies that $v \in V(H)$ and $v \in T$. Thus T' contains x.

Let $P = p_1p_2p_3p_4p_5$ be the subpath of H where $p_3 = v$. As T' contains x, p_2 and p_4 are contained in S. As (G, S, k) is reduced under Branching Rules 7.1 and 7.2, by Lemma 7.14, there is a bypassing vertex for P and v. Thus, $(G - T')[(V(H) \setminus \{v\}) \cup S]$ contains another DH obstruction. This contradicts the fact that $G - T$ is distance-hereditary, as $(G - T')[(V(H) \setminus \{v\}) \cup S]$ is an induced subgraph of $G - T$. □

We remark that when we move v into S in Reduction Rule 7.2, $k + \mathrm{cc}(G[S])$ does not increase, and the size of $V(G) \setminus S$ decreases. After applying Reduction Rule 7.2 exhaustively, we obtain that if the center of a star bag is unmarked, then this bag contains no unmarked leaves.

The next reduction rule arises directly from the definition of bypassing vertices.

Reduction Rule 7.3. Let v be a vertex in $G - S$ such that for every induced path $P = p_1p_2p_3p_4p_5$ with $p_3 = v$, there is a bypassing vertex for P and v. Then we remove v. In particular, when there is no such an induced path, we remove v.

Proof of soundness. This follows from Lemma 7.13. □

For fixed v, we can apply Reduction Rule 7.3 in time $\mathcal{O}(|V(G)|^5)$ by considering all vertex subsets of size 4, and testing whether p_2 and p_4 have a common neighbor in S.

We proceed by introducing a reduction rule which sequentially arranges bags containing exactly one twin class. The operation of swapping the adjacency between two vertices x and y in a graph is to remove xy if xy was an edge, and otherwise add an edge between x and y. The number of bags in D is strictly reduced when applying Reduction Rule 7.4.

Reduction Rule 7.4. Let B be a leaf bag and B' be the neighbor bag of B.

(1) See Figure 7.6. If B is a complete bag having exactly one twin class in $G - S$ and B' is a star bag whose leaf is adjacent to B, then we swap the adjacency between every two unmarked vertices in B. By swapping the adjacency, B becomes a star whose center is adjacent to B', and thus we can recompose the marked edge connecting B and B'. We recompose the marked edge connecting B and B'.

(2) See Figure 7.7. If B is a star bag having exactly one twin class in $G - S$, the center of B is adjacent to B', and B' is a complete bag, then we swap the adjacency between every two unmarked vertices in B. By swapping the adjacency, B becomes a complete graph, and thus we can recompose the marked edge connecting B and B'. We recompose the marked edge connecting B and B'.

The soundness of Reduction Rule 7.4 follows from the following lemma.

Lemma 7.15. *Let A be a set of vertices that are pairwise twins in G. Let G' be the graph obtained from G by swapping the adjacency relation between every pair of two distinct vertices in A. Then (G, S, k) is a* Yes*-instance if and only if (G', S, k) is a* Yes*-instance.*

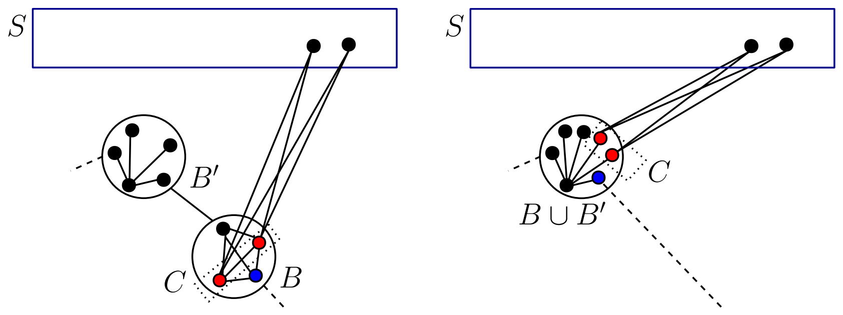


Figure 7.6: An illustration of Reduction Rule 7.4 Case 1. If C is the only twin class in B, we swap the adjacencies between vertices in C and recompose the marked edge between B and B'.

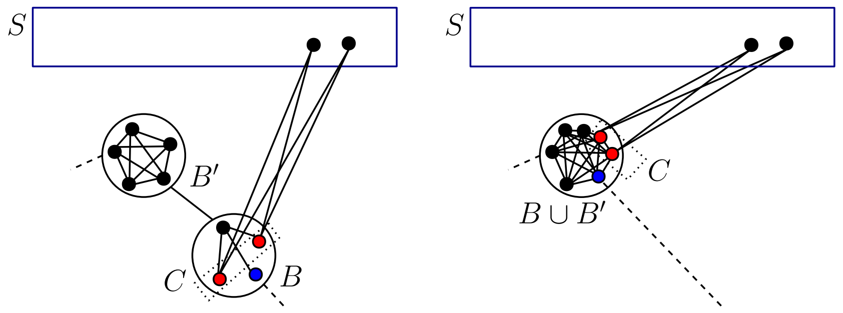


Figure 7.7: An illustration of Reduction Rule 7.4 Case 2.

Proof. Note that either $G[A]$ is a complete graph or it has no edges. Therefore, A is again a set of vertices that are pairwise twins in G'. Since each DH obstruction contains at most one vertex from a set of twins (and hence, at most one vertex from A), swapping the adjacency on A will neither introduce nor remove DH obstructions from G. Hence it is easy to check that (G, S, k) is a Yes-instance if and only if (G', S, k) is a Yes-instance. □

The last rule consider bags near to some leaf bag (see Figure 7.8 for an illustration). Recall that a star bag B is simple if its center is either unmarked or adjacent to a connected component of $D - V(B)$ consisting of one non-S-attached bag.

Reduction Rule 7.5. Let B_1, B_2, B_3 be distinct bags in D such that

- B_1 is a non-S-attached leaf bag whose neighbor bag is B_2, and it is not a star whose leaf is adjacent to B_2,
- B_2 has exactly two neighbor bags B_1 and B_3, it is a star whose center is adjacent to B_1, and the set of unmarked vertices in B_2 is the unique S-attached twin class C_2 in B_2, and

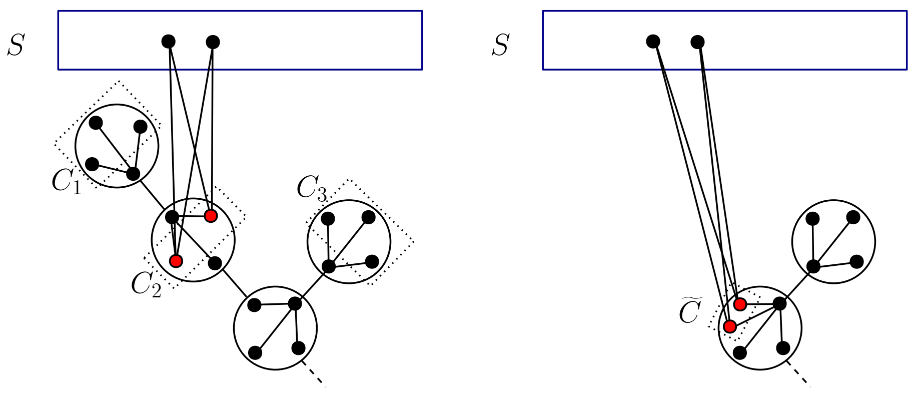


Figure 7.8: Reduction Rule 7.5.

- B_3 is a simple star bag.

Let C_1 be the set of unmarked vertices in B_1. Then we remove B_1 and B_2, and add a leaf set of unmarked vertices $\widetilde{C}$ with $\min(|C_1|, |C_2|)$ vertices to B_3, that is complete to $N_G(C_2) \cap S$ and has no other neighbors in S.

Note that this rule can potentially create an induced cycle of length 6. So, we need to run Branching Rule 7.1 after applying Reduction Rule 7.5.

Proof of soundness. As B_3 is a simple star bag and C_1 has no neighbors in S, $N_G(C_1) \setminus C_2$ is exactly the center of B_3 if it is unmarked, and otherwise, the set of unmarked vertices in the bag where the center of B_3 is adjacent. Let $C_3 = N_G(C_1) \setminus C_2$. We remark that C_3 is a twin class.

Let G' be the resulting graph obtained by applying Reduction Rule 7.5. Note that $\widetilde{C}$ is a set of pairwise twins in G' (it may not be a twin class), and $G - (C_1 \cup C_2) = G' - \widetilde{C}$.

We claim that (G, S, k) is a Yes-instance if and only if (G', S, k) is a Yes-instance. Suppose G has a minimum vertex set T such that $|T| \leq k$, $S \cap T = \emptyset$, and $G - T$ is distance-hereditary. We divide cases depending on whether T contains a vertex of $C_1 \cup C_2$ or not.

Case 1. ***T* contains a vertex in *$C_1 \cup C_2$*:** We observe that since C_i is a twin class and T is a minimum solution, if T contains a vertex of C_i, then T contains all vertices in C_i. Thus, T fully contains one of C_1 and C_2. Since $\widetilde{C} = \min(|C_1|, |C_2|)$, the set $T' = (T \setminus (C_1 \cup C_2)) \cup \widetilde{C}$ has size at most k. Moreover, we conclude that $G' - T'$ is distance-hereditary, as it is an induced subgraph of $G - T$.

Case 2. ***T* contains no vertex in *$C_1 \cup C_2$*:** Suppose that $G' - T$ contains a DH obstruction H. If H does not contain a vertex in $\widetilde{C}$, then H is an induced subgraph of

$G - T$, contradicting our assumption. Thus, H contains a vertex in $\widetilde{C}$, and as every pair of two distinct vertices in $\widetilde{C}$ is a twin, we have $|V(H) \cap \widetilde{C}| = 1$. Let v be the vertex in $V(H) \cap \widetilde{C}$, and let w, z be the two neighbors of v in H. As C_3 is a twin class in $G' - S$, at least one of w and z is contained in S. Without loss of generality, we assume $w \in S$.

If $z \in S$, then we can obtain a DH obstruction by replacing v with a vertex of C_2 in G, which implies that $G - T$ contains a DH obstruction. Thus, we may assume that z is contained in $V(G - S)$, and henceforth we have $z \in C_3$.

For two vertices $c_1 \in C_1$ and $c_2 \in C_2$, we can obtain a DH obstruction in $G - T$ from H by removing v and adding c_1, c_2, which is equivalent (up to isomorphism) to subdividing the unique edge in H incident to v and a vertex in C_3. By Observation 7.3, we know that the resulting graph $G - T$ must then also contain a DH obstruction, contradicting our assumption.

For the converse direction, suppose that G' has a minimum vertex set T' such that $|T'| \leq k$, $S \cap T' = \emptyset$, and $G' - T'$ is distance-hereditary. Similar to the forward direction, we divide cases depending on whether T' contains a vertex in $\widetilde{C}$ or not.

Case 1. T' contains no vertex in $\widetilde{C}$: Suppose $G - T'$ has a DH obstruction H. Since G has no small DH obstructions due to the application of branching rules, H should be an induced cycle of length at least 7. We have $V(H) \cap (C_1 \cup C_2) \neq \emptyset$, otherwise H is an induced subgraph of $G' - T'$, which is contradiction. As C_1 and C_2 are twin classes, H contains at most one vertex from each of C_1 and C_2.

We claim that H contains one vertex from each of C_1 and C_2. Suppose $V(H) \cap C_1 \neq \emptyset$ and $V(H) \cap C_2 = \emptyset$. Then the two neighbors of the vertex on $C_1 \cap V(H)$ belong to C_3, since $C_3 = N_G(C_1) \setminus C_2$. But C_3 forms a twin class, and an induced cycle of length at least 7 cannot contain two vertices from the same twin class; a contradiction. Suppose $V(H) \cap C_1 = \emptyset$ but $V(H) \cap C_2 \neq \emptyset$. Then the two neighbors of the vertex v in $V(H) \cap C_2$ in H are contained in S. Let $P = p_1p_2p_3p_4p_5$ be the subpath of H where $p_3 = v$. By Lemma 7.14, there is a bypassing vertex for P and v, and thus $G[(V(H) \setminus \{v\}) \cup S]$ contains a DH obstruction, which is also contained in $G' - T'$. This constitutes a contradiction. We conclude that H contains one vertex from each of C_1 and C_2.

It further implies that H contains one vertex from each of $C_3 = N_G(C_1) \setminus C_2$ and $N_G(C_2) \cap S$, because $N_G(C_2) \setminus C_1 \subseteq S$. Since H has length at least 7, we can obtain an induced cycle of length at least 6 in $G' - T$ from H by removing the vertices in $C_1 \cup C_2$ and adding one vertex of $\widetilde{C}$, which is contradiction.

Case 2. T' contains a vertex in $\widetilde{C}$: As $\widetilde{C}$ is a twin class and T' is a minimum solution for G', we have $\widetilde{C} \subseteq T'$. We obtain a set T from T' by removing $\widetilde{C}$, and adding C_1 if $|C_1| = |\widetilde{C}|$ and adding C_2 if $|C_2| = |\widetilde{C}|$. If $|C_1| = |C_2|$, then we add one of them chosen arbitrarily. Clearly, $|T| \leq |T'| \leq k$. We claim that $G - T$ is distance-hereditary.

In case when $C_2 \subseteq T$, we observe that every induced cycle of length at least 7 containing a vertex in C_1 has to contain two vertices in C_3, which is not possible. Thus, we may assume $C_1 \subseteq T$. Note that $N_G(C_2) \subseteq S \cup C_1$. Thus, whenever there is an induced cycle of length at least 7 in $G - T$ containing a vertex in C_2, by Lemma 7.14 there exists another DH obstruction which does not contain any vertex in C_2, contradicting the assumption that $G' - T' = G - (T \cup C_2)$ is distance-hereditary. Hence we conclude that $G - T$ is distance-hereditary. $\square$

Proposition 7.16. *Let (G, S, k) be an instance reduced under Branching Rules 7.1 and 7.2. Given a connected component H of $G - S$, we can in time $\mathcal{O}(|V(G)|^6)$ either apply one of Reduction Rules 7.1–7.5, or correctly answer that Reduction Rules 7.1–7.5 cannot be applied anymore.*

Proof. We first compute the canonical split decomposition D of H in time $\mathcal{O}(|V(G)| + |E(G)|)$ using Theorem 7.7. Then we classify twin classes in D by testing two unmarked vertices in a bag have the same neigbhorhood in S or not. This can be done in time $\mathcal{O}(|V(G)|^2)$. At the same time, we can also test whether a twin class is S-attached or not. Note that the total number of bags in canonical split decompositions of connected components of $G - S$ is $\mathcal{O}(|V(G)|)$.

We can apply Reduction Rules 7.1, 7.2, 7.4 in time $\mathcal{O}(|V(G)|)$, if one of them can be applied. We can apply Reduction Rule 7.3 in time $\mathcal{O}(|V(G)|^5)$ for fixed vertex v, and thus, we can test for all vertices $v \in V(G) \setminus S$ in time $\mathcal{O}(|V(G)|^6)$. For Reduction Rule 7.5, we need to consider three bags, which are uniquely identified by the first (leaf) bag among them, to check whether they satisfy preconditions of the rule. We can verify the preconditions of Reduction Rule 7.5 in constant time and thus this step takes time $\mathcal{O}(|V(G)|)$. We conclude that we can in time $\mathcal{O}(|V(G)|^6)$ either apply one of Reduction Rules 7.1–7.5, or correctly answer that Reduction Rules 7.1–7.5 cannot be applied anymore. $\square$

7.4.3 Structural Properties obtained after Exhaustive Application of Rules

In this subsection, we discuss structural properties obtained after the exhaustive application of both branching and reduction rules. We say that G and the canonical split decompositions of connected components of $G - S$ are reduced if Branching Rules 7.1–7.2 and Reduction Rules 7.1–7.5 cannot be applied anymore. We assume that the given instance is reduced in this subsection.

The following observation is a direct consequence of the exhaustive application of Reduction Rule 7.2.

Observation 7.17. *If the center of a star bag in D is unmarked, then this bag contains no unmarked leaves.*

Our next goal is to establish the following lemma.

Lemma 7.18. *Every bag of D contains at most one S-attached twin class.*

Before we formally prove Lemma 7.18, we briefly explain how the argument works. Let C_1 and C_2 be two distinct S-attached twin classes in a bag B such that neither of them consists of the center of a star and $(N_G(C_1) \setminus N_G(C_2)) \cap S$ is non-empty. If B is a star, then C_1 is anti-complete to C_2 and C_1, C_2 have a common neighbor in $G - S$, and thus, C_1 and C_2 satisfy preconditions of Lemma 7.11. Lemma 7.11 implies that $(N_G(C_1) \cap N_G(C_2)) \cap S$ is non-empty. Let $x \in (N_G(C_1) \setminus N_G(C_2)) \cap S$ and $y \in (N_G(C_1) \cap N_G(C_2)) \cap S$. We argue that whenever there is an induced path $w'wvzz'$ with $v \in C_1$ there is a bypassing vertex for v. We describe an example case. For instance, when w and z are both contained in $V(G - S)$, they are contained in $N_G(C_1) \cap N_G(C_2)$. So, $x \in N_G(C_1) \setminus N_G(C_2)$ while $w, z, y \in N_G(C_1) \cap N_G(C_2)$. Thus, by (2) of Lemma 7.11, if x is adjacent to y, then x should be adjacent to w and z, which means that x is a bypassing vertex for v. If x is not adjacent to y, then we could apply (3) of Lemma 7.11 to find a bypassing vertex for v. We do a careful analysis depending on the places of w and z, and also consider the case when B is a complete bag.

Proof of Lemma 7.18. Suppose there is a bag containing two distinct S-attached twin classes C_1 and C_2. By Observation 7.17, if C_i consists of the center of a star, then there are no other unmarked vertices in the bag, and thus it is not possible. Therefore, C_i does not consist of the center of a star bag. As C_1 and C_2 are distinct twin classes, $N_G(C_1) \cap S \neq N_G(C_2) \cap S$, and thus we have either $(N_G(C_1) \setminus N_G(C_2)) \cap S \neq \emptyset$ or $(N_G(C_2) \setminus N_G(C_1)) \cap S \neq \emptyset$. Without loss of generality, we assume $(N_G(C_1) \setminus N_G(C_2)) \cap S$ is non-empty.

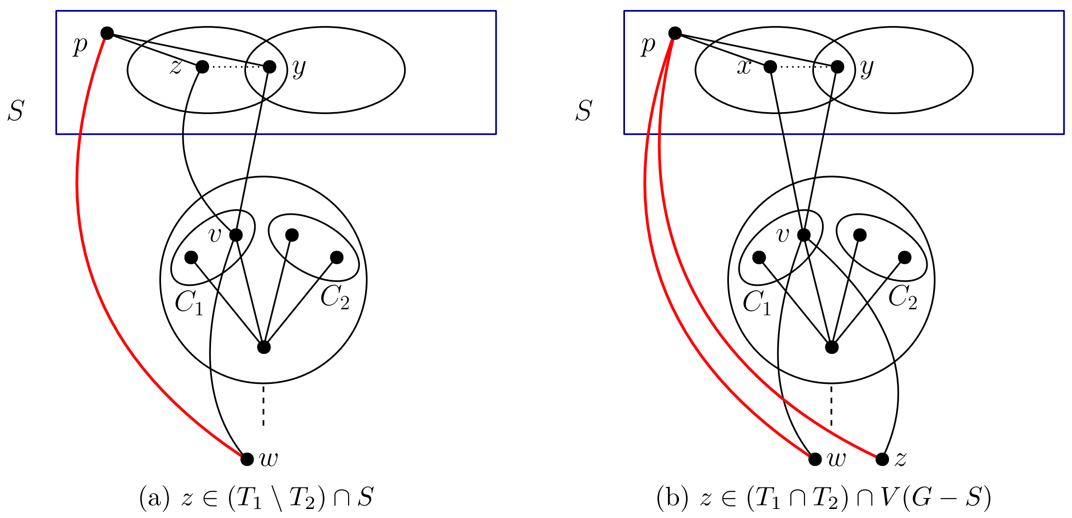
For each $i \in \{1, 2\}$, let $c_i \in C_i$ and let $T_i = N_G(C_i) \setminus C_{3-i}$. Let $x \in (T_1 \setminus T_2) \cap S$. We observe that $(T_1 \setminus T_2) \cap V(G - S) = (T_2 \setminus T_1) \cap V(G - S) = \emptyset$. This is because C_1 and C_2 are contained in B, which is a complete bag or a star bag whose center is marked.

We claim that for every $v \in C_1$ and every induced path $H = w'wvzz'$, there is a bypassing vertex for H and v. This will imply that we can apply Reduction Rule 7.3, which leads to a contradiction.

If $w, z \in S$, then by Lemma 7.14, there is a bypassing vertex for v. We may assume that w or z is contained in $G - S$. Without loss of generality, we assume that w is contained in $G - S$. We distinguish two cases depict in Figures 7.9 and 7.10.

Case 1. **B is a star bag:** In this case, C_1 is anti-complete to C_2 and $w \in (T_1 \cap T_2) \cap V(G - S)$. By (1) of Lemma 7.11, we have $(T_1 \cap T_2) \cap S \neq \emptyset$. Let $y \in (T_1 \cap T_2) \cap S$. We divide into two cases depending on whether $z \in (T_1 \setminus T_2) \cap S$ or $z \in T_1 \cap T_2$.

Suppose $z \in (T_1 \setminus T_2) \cap S$. Note that both y and w are contained in $T_1 \cap T_2$. Since $zw \notin E(G)$, by (2) of Lemma 7.11, z is not adjacent to y. Since y and z are neighbors of v and y, z are not adjacent, by (2) of Lemma 7.10, there is an induced path zpy for some



S
p
z
y
v
C_1
C_2
w
(a) $z \in (T_1 \setminus T_2) \cap S$
S
p
x
y
v
C_1
C_2
w
z
(b) $z \in (T_1 \cap T_2) \cap V(G - S)$

Figure 7.9: When B is a star bag in Lemma 7.18. The red thick edges illustrate the edges whose existence is guaranteed by Lemma 7.11.

$p \in S$. Then by (3) of Lemma 7.11, p is adjacent to w, and therefore, p is a bypassing vertex.

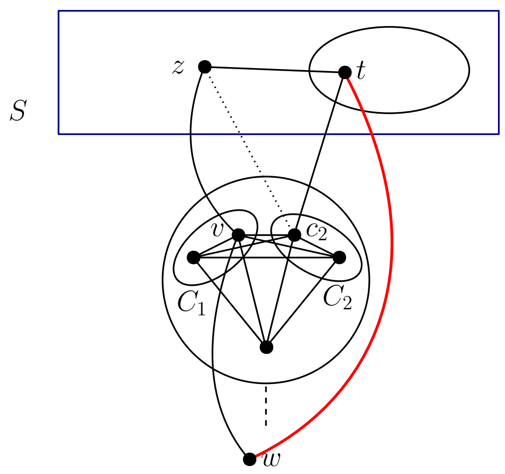
Suppose $z \in T_1 \cap T_2$. Recall that x is a vertex in $(T_1 \setminus T_2) \cap S$. If x is adjacent to y, then by (2) of Lemma 7.11, x is adjacent to both w and z, and thus x is a bypassing vertex. We may assume that $xy \notin E(G)$. Then by (2) of Lemma 7.10, there is an induced path xpy for some $p \in S$. By (3) of Lemma 7.11, p is adjacent to both w and z, and therefore, p is a bypassing vertex, as required.

Case 2. **B is a complete bag:** Note that C_1 is complete to C_2, and w is contained in either C_2 or $(T_1 \cap T_2) \cap V(G - S)$. We first discuss when w is contained in C_2.

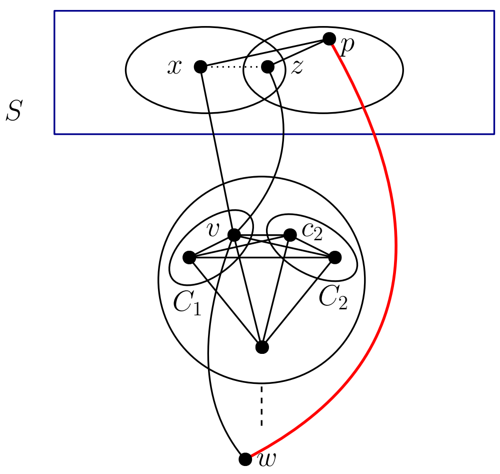
Suppose $w \in C_2$. As w is not adjacent to z, z cannot be in $T_1 \cap T_2$, and furthermore z cannot be in B as B is a complete bag. Thus $z \in (T_1 \setminus T_2) \cap S$. If z has a neighbor in $T_2 \cap S$, then the neighbor is a bypassing vertex, because it is adjacent to both w and z. We may assume that z has no neighbors in $T_2 \cap S$. Observe that z and $T_2 \cap S$ are contained in the same connected component of $G[S]$, otherwise, Branching Rule 7.2 can be applied. Let us take a shortest path P from z to $T_2 \cap S$ in $G[S]$. Then Pw is an induced path of length at least 3 and v is adjacent to its end vertices, and thus $G[S \cup \{v, w\}]$ has a DH obstruction by Lemma 7.4, which contradicts the assumption that (G, S, k) is reduced under Branching Rule 7.1.

Now, suppose $w \in (T_1 \cap T_2) \cap V(G - S)$. In this case, z is not in C_2. We distinguish subcases by the placement of z. We illustrate cases in Figure 7.10.

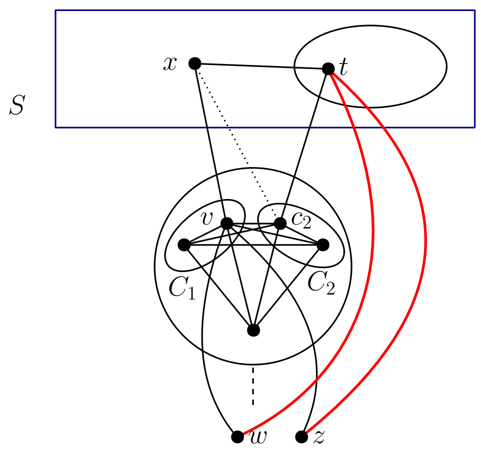
Case 2-1. $z \in (T_1 \setminus T_2) \cap S$ **:** Let P be a shortest path from z to $T_2 \cap S$. If P has length at least 2, then Pc_2 is an induced path of length at least 3 and v is adjacent to



(a) $z \in (T_1 \setminus T_2) \cap S$.



(b) $z \in (T_1 \cap T_2) \cap S$.



(c) $z \in (T_1 \cap T_2) \cap V(G - S)$.

Figure 7.10: When B is a complete bag and $w \in (T_1 \cap T_2) \cap V(G - S)$ in Lemma 7.18. The red thick edges illustrate the edges whose existence is guaranteed by Lemma 7.12 or non-existence of small DH obstructions.

its end vertices. So, $G[S \cup \{v, c_2\}]$ contains a DH obstruction, which is a contradiction. Thus, z has a neighbor in $T_2 \cap S$, say t. If w is not adjacent to t, then ztc_2w is an induced path and v is adjacent to its end vertices. This contradicts the assumption that (G, S, k) is reduced under Branching Rule 7.1. Thus, $wt \in E(G)$ and t is a bypassing vertex.

Case 2-2. $z \in (T_1 \cap T_2) \cap S$ **:** Recall that x is a vertex in $(T_1 \setminus T_2) \cap S$. If x is adjacent to z, then by (1) of Lemma 7.12, w is adjacent to x because w is not adjacent to z. Thus, x is a bypassing vertex. So, we may assume that x is not adjacent to z. By (2) of Lemma 7.10, there is an induced path xpz for some $p \in S$.

If $p \in (T_1 \setminus T_2) \cap S$, then p is adjacent to w by (1) of Lemma 7.12, and thus p is a bypassing vertex. Assume $p \in (T_1 \cap T_2) \cap S$. If p is adjacent to w, then p is a bypassing vertex, and we are done. Otherwise, by (1) of Lemma 7.12, x should be adjacent to both w and z, since $wz \notin E(G)$. Therefore, we may assume that $p \notin T_1$. Then by (2) of Lemma 7.12, we have $p \in (T_2 \setminus T_1) \cap S$, and again by (1) of Lemma 7.12, either $wz \in E(G)$ or $wp \in E(G)$. Since $wz \notin E(G)$, p becomes a bypassing vertex.

Case 2-3. $z \in (T_1 \cap T_2) \cap V(G - S)$ **:** If x is adjacent to w or z, then by (1) of Lemma 7.12, x is adjacent to both w and z, because $wz \notin E(G)$. Then x is a bypassing vertex. Therefore, we may assume x is adjacent to neither w nor z. We take a shortest path P from x to $T_2 \cap S$. If P has length at least 2, then Pc_2 is an induced path of length at least 3, and since v is adjacent to its end vertices, $G[V(P) \cup \{v, c_2\}]$ contains a DH obstruction by Lemma 7.4. But this contradicts the assumption that G is reduced under Branching Rule 7.1. We may assume that P has length 1, and let t be a neighbor of x in $T_2 \cap S$. Observe that if t is not adjacent to w or z, then xtc_2w or xtc_2z is an induced path, respectively, and v is adjacent to its end vertices. This contradicts the assumption that (G, S, k) is reduced under Branching Rule 7.1. Therefore t is adjacent to both w and z, which implies that t is a bypassing vertex.

We conclude that, for every induced path $w'wvzz'$, there exists a bypassing vertex for v. This contradicts the assumption that D is reduced under Reduction Rule 7.3. Therefore, every bag of D contains at most one S-attached twin class. □

Lemma 7.19. *Let B be a bag and D_1 be a connected component of $D - V(B)$ containing no S-attached bags. Then B is a simple star bag whose center is adjacent to D_1.*

Proof. Let B_1 be the neighbor bag of B contained in D_1. First claim that $D_1 = B_1$. Suppose D_1 contains at least one bag other than B_1. We regard B_1 as the root bag of D_1, and choose a bag Y in D_1 with maximum $|\text{path}(Y, B_1)|$. Clearly, Y is a leaf bag in D. Let X be the neighbor bag of Y.

Suppose X is a star. As we choose Y with maximum $|\text{path}(Y, B_1)|$, every child of X is a leaf bag. We claim that there is no leaf bag of D pending to a leaf of X. Suppose for contradiction there exists such a bag Y_1. Since D is canonical, Y_1 is not a star whose center is adjacent to X. If Y_1 is a star whose leaf is adjacent to X, then it can be reduced under Reduction Rule 7.2. If Y_1 is a complete graph, then it can be reduced under Reduction Rule 7.4, which is a contradiction. Therefore, there is no leaf bag of D pending to a leaf of X, and it implies that the center of X is adjacent to Y, and Y is the unique child of X.

Let v be an unmarked vertex of X. As $N_G(v)$ is the set of unmarked vertices in Y which is a twin class, any induced path of length 4 could not contain v as the third vertex. Therefore, Reduction Rule 7.3 can be applied to remove v, which is a contradiction.

Suppose X is a complete graph. Since D is canonical, Y is not a complete graph. If Y is a star whose leaf is adjacent to X, then all unmarked leaves in Y can be removed by

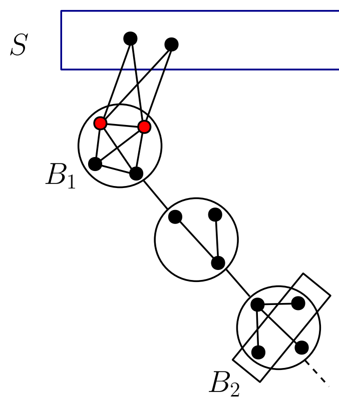


Figure 7.11: Illustration for the case treated in Lemma 7.20.

Reduction Rule 7.2. If Y is a star whose center is adjacent to X, then we can apply Reduction Rule 7.4 to X and Y.

We conclude that $D_1 = B_1$. Moreover, if B is not a star whose center is adjacent to B_1, then we can reduce B_1 using Reduction Rule 7.2 or 7.4. Thus B is a star whose center is adjacent to B_1. $\square$

The following structure is illustrated in Figure 7.11.

Lemma 7.20. *Let B_1 be a leaf bag containing at most one S-attached twin class and B_2 be a bag distinct from B_1 such that*

- *B_2 is a star bag whose center is adjacent to* $\text{comp}(B_2, B_1)$.
- *every bag in* $\text{path}(B_1, B_2)\setminus\{B_1, B_2\}$ *is not a (B_1, B_2)-separator bag, and has exactly two neighbor bags, and*
- *for every bag B in* $\text{path}(B_1, B_2)\setminus\{B_1, B_2\}$ *that is not a star whose center is adjacent to* $\text{comp}(B, B_1)$*, B is non-S-attached.*

Then B_2 contains no non-S-attached twin class C.

Proof. Suppose B_2 contains a non-S-attached twin class, and let v be a vertex in the class. We claim that there is no induced path $w'wvzz'$, which implies that Reduction Rule 7.3 can be applied. Suppose there is such a path.

We claim that either $N_G(w) \subseteq N_G(z)$ or $N_G(z) \subseteq N_G(w)$. If $N_G(w) \subseteq N_G(z)$ then w' should be adjacent to z, which contradicts the fact that $w'wvzz'$ is an induced path. The same argument holds when $N_G(z) \subseteq N_G(w)$.

Let P_w and P_z be the bags containing w and z, respectively. As B_2 is a star whose center is adjacent to $\text{comp}(B_2, B_1)$, P_w and P_z are bags in $\text{path}(B_1, B_2) \setminus \{B_2\}$.

First assume $P_w = P_z = B_1$. In this case, since w is not adjacent to z, B_1 is a star bag. Note that no DH obstruction contains two twins, and therefore, w and z are contained in distinct twin classes. Since B_1 contains at most one S-attached twin class by Lemma 7.18, one of w and z is contained in the non-S-attached twin class in B_1. Say w is such a vertex. Then we have $N_G(w) \subseteq N_G(z)$, because w and z are twins in $G - S$.

Now, we assume at least one of P_w and P_z is not equal to B_1. We further assume P_z is contained in $\text{path}(P_w, B_2) \setminus \{B_2\}$. The same argument holds when P_w is contained in $\text{path}(P_z, B_2) \setminus \{B_2\}$.

Since $P_z \in \text{path}(P_w, B_2) \setminus \{B_2\}$ and $wz \notin E(G)$, P_z is not a complete bag. Thus, P_z is a star bag whose center is adjacent to $\text{comp}(P_z, B_2)$. As $P_z \neq B_1$ and it is not S-attached by the assumption, all neighbors of z in G are neighbors of w. Then z' should be adjacent to w, which is a contradiction.

We conclude that there is no such path $w'wvzz'$, and Reduction Rule 7.3 can be applied to remove v. Therefore, B_2 contains no non-S-attached twin class C. ☐

The following structure is illustrated in Figure 7.12.

Lemma 7.21. *Let B_1 be a leaf bag having exactly one S-attached twin class and B_2 be a simple star bag distinct from B_1 such that*

- *B_1 is not a star whose leaf is adjacent to a neighbor bag,*
- *every bag in $\text{path}(B_1, B_2) \setminus \{B_1, B_2\}$ is non-S-attached, not a (B_1, B_2)-separator bag and has exactly two neighbor bags.*

Then B_1 contains no non-S-attached twin class.

Proof. We claim that if B_1 contains a non-S-attached twin class, then we can apply a reduction rule to remove it. Suppose B_1 contains a non-S-attached twin class C_1, and let C_2 be the S-attached twin class in B_1.

Let $v \in C_1$ and we claim that there is no induced path $w'wvzz'$. If this is true, then we can apply Reduction Rule 7.3. Suppose there is such an induced path.

Assume that $w \in V(B_1)$. In this case, B_1 should be a complete bag. Therefore, z is adjacent to w, because $z \in V(G - S)$, and w, v are twins in $G - S$. This contradicts the assumption that $w'wvzz'$ is an induced path. Thus, we can assume that $w \notin V(B_1)$, and similarly, $z \notin V(B_1)$.

By symmetry, we assume $|\text{path}(B_1, B_w)| \leq |\text{path}(B_1, B_z)|$, where B_w and B_z are bags containing w and z, respectively. Since w is not adjacent to z, B_w should be a star bag whose center is adjacent to the component $\text{comp}(B_w, B_1)$. Therefore, every neighbor of w in $G - S$ is adjacent to z, and in particular, w' is adjacent to z. This contradicts the assumption that $w'wvzz'$ is induced. This proves the claim. This contradicts the assumption that D is reduced. ☐

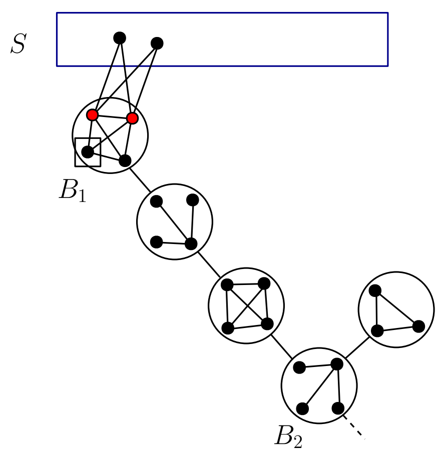


Figure 7.12: Illustration for the case treated in Lemma 7.21.

Lemma 7.22. *Let B be a simple star bag, and let D_1 be a connected component of $D - V(B)$ such that*

- *D_1 contains exactly one S-attached bag B_1, and*
- *there is no (B_1, B)-separator bag.*

Then B_1 is a star whose leaf is adjacent to $\text{comp}(B_1, B)$ *and there is a leaf bag B_2 where the center of B_1 is adjacent to B_2.*

Proof. We first claim that B_1 is a star whose leaf is adjacent to $\text{comp}(B_1, B)$. We prove this by a sequence of auxiliary claims. Suppose for contradiction that B_1 does not satisfy the property; that is, either B_1 is a complete bag or a star bag whose center is adjacent to $\text{comp}(B_1, B)$.

Claim 7.23. *There is no connected component of $D - V(B_1)$ other than* $\text{comp}(B_1, B)$.

Proof of the Claim. If there is such a component C_1, then by the assumption, it contains no S-attached bag. By Lemma 7.19, B_1 is a star whose center is adjacent to C_1, contradicting our assumption. Thus, there is no connected component of $D - V(B_1)$ other than $\text{comp}(B_1, B)$. ♢

We observe that B_1 contains one S-attached twin class by Lemma 7.18. Also, all bags in $\text{path}(B, B_1) \setminus \{B, B_1\}$ have exactly two neighbor bags. This follows from Lemma 7.19 and the fact that every bag in $\text{path}(B_1, B) \setminus \{B, B_1\}$ is not a (B, B_1)-separator bag. Now, we can observe that B and B_1 satisfy the conditions of Lemma 7.21. Therefore, B_1 contains no non-S-attached twin class.

Claim 7.24. *There is no star bag $B_2 \in \text{path}(B, B_1) \setminus \{B, B_1\}$ whose center is adjacent to* $\text{comp}(B_2, B_1)$.

Proof of the Claim. Suppose there is such a star bag B_2. Then B_1 and B_2 satisfy the conditions in Lemma 7.20. Thus, B_2 has no non-S-attached twin class. But this is impossible as B_2 has only two neighbor bags and B_2 has no S-attached twin class. $\Diamond$

By Claim 7.24, we observe that B_1 and its parent bag satisfy the condition (1) or (2) of Reduction Rule 7.4, and thus we can apply the rule. This contradicts the assumption that D is reduced. Thus, B_1 is a star whose leaf is adjacent to $\text{comp}(B_1, B)$.

Now, suppose there is no bag B_2 where the center of B_1 is adjacent to B_2. Since there is no bag pending to a leaf of B_1 by Lemma 7.19, B_1 is a leaf bag. In this case, we can reduce using Reduction Rule 7.2, which is a contradiction. Therefore, there is a leaf bag B_2 where the center of B_1 is adjacent to B_2, as required. $\square$

Lemma 7.25. *Let B_1 and B_2 be two simple star bags in D such that*

- *every bag in* $\text{path}(B_1, B_2) \setminus \{B_1, B_2\}$ *is a non-S-attached bag, has two neighbor bags, and is not a (B_1, B_2)-separator bag.*

Then B_1 and B_2 are neighbor bags.

Proof. Suppose for contradiction that $\text{path}(B_1, B_2) \setminus \{B_1, B_2\} \neq \emptyset$. Let B be a bag in $\text{path}(B_1, B_2) \setminus \{B_1, B_2\}$ and v be an unmarked vertex of B.

We claim that there is no induced path $w'wvzz'$. Suppose there is such an induced path. By symmetry, we assume $|\text{path}(B_1, B_w)| \leq |\text{path}(B_1, B_z)|$, where B_w and B_z are bags containing w and z, respectively. First, assume that w and z are contained in the different connected components of $D - V(B)$. Since B is the not (B_1, B_2)-separator bag, v cannot be a center of a star bag. Hence, the only possibility for v to be a neighbor of both w and z is if B is a complete bag. But then w is adjacent to z, contradiction. Thus, w and z are contained in the same connected component of $D - V(B)$. Without loss of generality, we may assume that such a connected component contains B_1.

Suppose there is a bag B_1' where the center of B_1 is adjacent to B_1'. Since B_1 is simple, B_1' contains only one non-S-attached twin class. Thus one of w and z are not contained in B_1', as they are not twins in the path $w'wvzz'$.

We may assume w is in $\text{path}(B_1, B) \setminus \{B_1, B\}$. Then, every neighbor of w in $G - S$ is adjacent to z, in particular, w' is adjacent to z. This contradicts the assumption that $w'wvzz'$ is induced.

This proves the claim. Since there is no such a path $w'wvzz'$, we can apply Reduction Rule 7.3 to remove v. This contradicts the assumption that D is reduced. We conclude that B_1 and B_2 are neighbor bags. $\square$

Finally, we claim that our instance has the desired inseparability property. We formalize and prove this property below.

Lemma 7.26. *Let B be a bag and let D_1 be a connected component of $D - V(B)$ such that D_1 contains exactly one S-attached bag B_1. Then there is no (B_1, B)-separator bag.*

Proof. For contradiction, suppose that there is a (B_1, B)-separator bag. We choose such a bag B_2 with minimum $|\text{path}(B_1, B_2)|$. From the choice of B_2, there is no (B_1, B_2)-separator bag.

We verify preconditions of Lemma 7.22 for B_1 and B_2. Clearly, $\text{comp}(B_2, B_1)$ has exactly one S-attached bag B_1, and there is no (B_1, B_2)-separator bag. To see that B_2 is a simple star bag, let us assume that there is a connected component D_2 of $D - V(B_2)$ where the center of B_2 is adjacent to D_2; if there is no such a component, it is clear by definition. As D_2 contains no S-attached bag, by Lemma 7.19, D_2 consists of one bag and B_2 is a simple star bag.

By applying Lemma 7.22 for B_1 and B_2, we can observe that B_1 is a star whose leaf is adjacent to $\text{comp}(B_1, B_2)$, and there is a leaf bag B_ℓ where the center of B_1 is adjacent to B_ℓ. We can also observe that B_1 is a simple star bag. By Lemma 7.19, there is no connected component of $D - V(B_1)$ pending to leaves of B_1 other than the leaf adjacent to its parent.

Note that every bag A in $\text{path}(B_1, B_2) \setminus \{B_1, B_2\}$ has two neighbor bags, because it is not a (B_1, B_2)-separator bag and by Lemma 7.19 there is no other component $D - V(A)$ pending to A. Therefore by Lemma 7.25, B_2 is a neighbor bag of B_1.

Now, by Lemma 7.20, there is no non-S-attached twin class in B_1, which means that the unmarked vertices of B_1 form a unique S-attached twin class. Then, we can apply Reduction Rule 7.5 to B_ℓ, B_1, B_2, a contradiction. □

7.4.4 Connected components with two S-attached bags

This section is devoted to showing that if D is reduced and contains two distinct S-attached classes, then we can apply a reduction rule. Suppose D is reduced and contains two distinct S-attached classes, and we choose a root bag of D. Let B be a farthest bag from the root bag such that there are two descendant bags B_1 and B_2 of B having distinct S-attached twin classes C_1 and C_2, respectively.

First, we verify that the distance from C_1 to C_2 in $G - S$ is at most 2.

Lemma 7.27. *The distance from C_1 to C_2 in $G - S$ is at most* 2.

Proof. Let us take a shortest sequence of twin classes $(C_1 = U_0) - U_1 - \cdots - U_t - (C_2 = U_{t+1})$ from C_1 to C_2 in $G - S$ such that for $i, j \in \{0, 1, \ldots, t+1\}$ with $i \neq j$, U_i is complete to U_j if $|i - j| = 1$ and they are anti-complete, otherwise. We note that each U_i except U_0 and U_{t+1} corresponds to a

(B_1, B_2)-separator bag. Clearly, at most one of $U_1, \dots, U_t$ possibly has a neighbor in S because C_i is the unique S-attached twin class in $\text{comp}(B, B_i)$ if $B_i \neq B$. By (3) and (4) of Lemma 7.10, the length from C_1 to C_2 in $G - S$ cannot be 3 or 4, and thus t cannot be 2 or 3. Also, by Lemma 7.26, we know that there is no (B_i, B)-separator bag when $B_i \neq B$. Thus, t cannot be larger than 3. So, the distance from C_1 to C_2 in $G - S$ is at most 2. □

Proposition 7.28. *The bag B is not a (C_1, C_2)-separator bag.*

Proof. For each $i \in \{1, 2\}$ let $T_i = N_G(C_i)$. Since by Lemma 7.27 the distance from C_1 to C_2 is at most 2, it follows from Observation 7.9 that there exists at most one (C_1, C_2)-separator bag. Suppose that B is the (C_1, C_2)-separator bag. Note that $B_i \neq B$ for some $i \in \{1, 2\}$ because B_1 and B_2 are distinct. Without loss of generality, we assume that $B_1 \neq B$. We verify the proposition by a sequence of claims.

Claim 7.29. *B_1 is not a star bag whose leaf is adjacent to* $\text{comp}(B_1, B)$.

> *Proof of the Claim.* Suppose B_1 is a star bag whose leaf is adjacent to $\text{comp}(B_1, B)$. As B_1 is the unique S-attached bag in $\text{comp}(B, B_1)$, by Lemma 7.19, there is no bag pending to a leaf of B_1. Also, the center of B_1 is marked, otherwise, we can apply Reduction Rule 7.2, and by Lemma 7.19, B_1 is a simple star bag. Therefore, C_1 consists of leaves of B_1, and B_1 is a (C_1, C_2)-separator bag. But this contradicts the assumption that $B \neq B_1$ and B is the only (C_1, C_2)-separator bag. ♢

Note that B_1 is either a complete graph, or a star whose center is adjacent to $\text{comp}(B_1, B)$. We observe that B_1 contains a non-S-attached twin class.

Claim 7.30. *B_1 contains a non-S-attached twin class.*

> *Proof of the Claim.* Suppose for contradiction that B_1 contains no non-S-attached twin class, that is, C_1 is exactly the set of unmarked vertices of B_1. Let B_3 be the parent bag of B_1. If B_3 is not a star whose center is adjacent to B_1, then we can apply Reduction Rule 7.4. We may assume B_3 is a star whose center is adjacent to B_1. But in this case, $B \neq B_3$, and thus, B_3 has no S-attached twin classes. By Lemma 7.19, B_3 has exactly two neighbor bags, and by Lemma 7.20, it contains no non-S-attached twin class. But this is impossible. We conclude that B_1 contains a non-S-attached twin class. ♢

Claim 7.31. *There is a vertex x in $(T_1 \setminus T_2) \cap V(G - S)$ contained in a complete bag such that x has no neighbors in S.*

Proof of the Claim. If B_1 is a complete bag, then the non-S-attached twin class is contained in $(T_1 \setminus T_2) \cap V(G-S)$. Assume B_1 is a star. Since B is a star whose leaf is adjacent to $\text{comp}(B, B_1)$, there is at least one bag in $\text{path}(B_1, B) \setminus \{B_1, B\}$. Moreover, there is no star bag B' in $\text{path}(B_1, B) \setminus \{B_1, B\}$ whose center is adjacent to $\text{comp}(B', B_1)$ by Lemma 7.20. Therefore, there is at least one complete bag in $\text{path}(B_1, B) \setminus \{B_1, B\}$, which contains a vertex in $(T_1 \setminus T_2) \cap V(G-S)$. We choose x to be such a vertex. Then x is a vertex in $(T_1 \setminus T_2) \cap V(G-S)$ having no neighbors in S, and also contained in a complete bag. ♢

Since x is contained in a complete bag, x has a neighbor in $(T_1 \cap T_2) \cap V(G-S)$. By (1) of Lemma 7.11, we have $(T_1 \cap T_2) \cap S \neq \emptyset$. Since $x \in T_1 \setminus T_2$ and x is adjacent to a vertex in $T_1 \cap T_2$, by (2) of Lemma 7.11, x should be adjacent to all vertices in $(T_1 \cap T_2) \cap S$, which contradicts the fact that x has no neighbors in S. □

The following lemma describes all possible cases.

Lemma 7.32. *Let B be a farthest bag from the root bag such that there are two descendant bags B_1 and B_2 of B having distinct S-attached twin classes C_1 and C_2, respectively. Then $B_1 \neq B_2$ and one of the following happens:*

1. *The distance from C_1 to C_2 in $G-S$ is 2 and the unique (C_1, C_2)-separator bag is contained in* $\text{comp}(B, B_i)$ *for some $i \in \{1, 2\}$.*
2. *C_1 is complete to C_2 and either*
   - *B is a star bag and C_i is the set consisting of the center of B for some $i \in \{1, 2\}$, or*
   - *B is a star bag whose center is adjacent to* $\text{comp}(B, B_i)$ *for some $i \in \{1, 2\}$.*
3. *C_1 is complete to C_2 and B is a complete bag.*

Proof. By Lemma 7.18, each bag contains at most one S-attached twin class and it follows that $B_1 \neq B_2$. By Lemma 7.27 the distance from C_1 to C_2 in $G-S$ is at most 2. Suppose that the distance from C_1 to C_2 in $G-S$ is 2. Then, there is a unique (C_1, C_2)-separator bag in D. By Proposition 7.28, B cannot be the (C_1, C_2)-separator bag. Thus, the unique (C_1, C_2)-separator bag in contained in $\text{comp}(B, B_i)$ for some $i \in \{1, 2\}$. If the distance from C_1 to C_2 is 1, then C_1 is complete to C_2, and in this case if B is a star, then its center either consists of one class C_i or is adjacent to one of $\text{comp}(B, B_1)$ and $\text{comp}(B, B_2)$. □

We show that in each of three cases in Lemma 7.32, we can apply a reduction rule.

Proposition 7.33. *Suppose the distance from C_1 to C_2 in $G - S$ is 2 and the unique (C_1, C_2)-separator bag is contained in* $\text{comp}(B, B_2)$*. Then for every induced path $P = p_1p_2p_3p_4p_5$ with $p_3 \in C_2$, there is a bypassing vertex for P and p_3.*

Proof. Let $T_i = N_G(C_i)$ for each $i \in \{1, 2\}$. We start by proving the following claim.

Claim 7.34. *The bag B_2 is the (C_1, C_2)-separator bag.*

> *Proof of the Claim.* For a contradiction, suppose that B_2 is not the (C_1, C_2)-separator bag and let $B' \neq B_2$ be the (C_1, C_2)-separator bag. Then B' is a (B_2, B)-separator bag. However, since $\text{comp}(B, B_2)$ has exactly one S-attached bag B_2, by Lemma 7.26, there is no (B_2, B)-separator bag, which is a contradiction. $\diamondsuit$

By Lemma 7.19, B_2 has no child pending to a leaf of B_2. If the center of B_2 is unmarked, then we can reduce it using Reduction Rule 7.2. Thus there is component attached to the center of B_2, and by Lemma 7.19 this component is a single leaf bag. We call the leaf bag B_3, and let C_3 be the set of unmarked vertices of B_3. Note that C_3 is a non-S-attached twin class. Also, by Lemma 7.20, B_2 contains no non-S-attached twin class.

Suppose there is an induced path $P = p_1p_2p_3p_4p_5$ with $p_3 \in C_2$. We want to show that there is a bypassing vertex for P and p_3. Observe that every neighbor of p_3 in G is either in S or in C_3. As C_3 is a twin class, it contains at most one of p_2 and p_4. If p_2 and p_4 are contained in S, then by Lemma 7.14, there is a bypassing vertex for p_3. Thus, we may assume that one of p_2 and p_4 is contained in S and the other is contained in C_3.

By symmetry, we may assume that $p_2 \in C_3$ and $p_4 \in S$. Note that since $p_2 \in C_3$, p_2 has no neighbors in S. Furthermore, as the distance from C_1 to C_2 is exactly 2, C_3 is complete to C_1. This implies that $p_2 \in (T_1 \cap T_2) \cap V(G - S)$.

By (1) of Lemma 7.11, we have $(T_1 \cap T_2) \cap S \neq \emptyset$. Let $t \in (T_1 \cap T_2) \cap S$. We divide cases depending on whether p_4 is in $T_2 \setminus T_1$ or $T_1 \cap T_2$.

Case 1. $p_4 \in (T_2 \setminus T_1) \cap S$**:** Note that $p_4 \in T_2 \setminus T_1$ and $p_2, t \in T_1 \cap T_2$. Since p_4 is not adjacent to p_2, by (2) of Lemma 7.11, p_4 is not adjacent to t as well. As p_4 and t are neighbors of p_3 and (G, S, k) is reduced under Branching Rule 7.2, p_4 and t are contained in the same connected component of $G[S]$. Moreover, since (G, S, k) is reduced under Branching Rule 7.1, there is no induced path of length at least 3 from p_4 to t in $G[S]$, and thus the distance from p_4 to t in $G[S]$ is exactly 2. Let p_4pt be an induced path for some $p \in S$.

If p is contained in $T_1 \cap T_2$, then p_4 should be adjacent to p_2 by (2) of Lemma 7.11. Thus, p is not contained in $T_1 \cap T_2$. If $p \in T_2 \setminus T_1$, then by (2) of Lemma 7.11, p is adjacent to p_2, but p_2 has no neighbors in S, a contradiction. Lastly, assume that $p \in S \setminus T_2$. In this case, by (3) of Lemma 7.11 with $(p, x, y_1, y_2) = (p, p_4, t, p_2)$, p is adjacent to p_2, again a contradiction.

Case 2. $p_4 \in (T_1 \cap T_2) \cap S$**:** Let $c_1 \in C_1$. If p_2 and c_1 have a common neighbor c in $G - S$ that is adjacent to neither p_3 nor p_4, then $G[\{p_2, p_3, p_4, c, c_1\}]$ is isomorphic to the house. So, there are no such vertices. This implies that for each $i \in \{1, 2\}$, there is no complete bag in $\text{path}(B_i, B) \setminus \{B, B_i\}$, and if B_1 or B is a complete bag, then it contains no non-S-attached twin class.

Claim 7.35. $\text{path}(B, B_1)$ *contains at most* 3 *bags, and when it contains* 3 *bags, the bag in* $\text{path}(B, B_1)$ *is a star bag whose center is adjacent to* B*.*

Proof of the Claim. Suppose $\text{path}(B, B_1)$ contains more than 3 bags, and let B_1' be the parent bag of B_1 and B_1'' be the parent of B_1'. As $\text{path}(B, B_1) \setminus \{B\}$ contains no complete bags, both B_1' and B_1'' are star bags. Thus, B_1'' is a star bag whose center is adjacent to B_1'. Such a bag B_1'' does not exist by Lemma 7.20. ♢

In particular, Claim 7.35 implies that every neighbor of C_3 is either in C_2 or not contained in the component of $D - V(B)$ containing B_2.

We divide into subcases depending on the shape of B_1.

Case 2-1. B_1 **is a complete bag:** First assume that $B_1 = B$. As B_1 contains no non-S-attached twin class and $p_1p_4 \notin E(G)$, p_1 is in the neighborhood of C_1 in $G - S$. Then c_1 is adjacent to end vertices of $p_1p_2p_3p_4$, and by Lemma 7.4, G contains a small DH obstruction. This is a contradiction. We may assume $B_1 \neq B$.

As D is canonical, the parent bag B_1' of B_1 is a star bag. We claim that $B = B_1'$. Suppose $B \neq B_1'$, that is, B_1' is contained in $\text{path}(B, B_1) \setminus \{B, B_1\}$. Since there is no (B, B_1)-separator bag, the center of B_1' is adjacent to either $\text{comp}(B_1', B)$ or B_1. As B_1 is the unique S-attached bag in $\text{comp}(B, B_1)$, by Lemma 7.19, B_1' has exactly two neighbor bags. Also, again by Lemma 7.19, B_1 is a leaf bag. Therefore, by Lemma 7.20, the center of B_1' is not adjacent to B_1. On the other hand, if the center of B_1' is adjacent to $\text{comp}(B_1', B_1)$, then we can apply Reduction Rule 7.4, as B_1 contains no non-S-attached twin class, which is a contradiction. Thus, we have $B = B_1'$, and the same argument using Reduction Rule 7.4 implies that the center of B_1' is adjacent to B. Then p_1 should be contained in B_1 and adjacent to p_4, which is impossible.

Case 2-2. B_1 **is a star bag:** First assume that $B_1 = B$. As C_1 is complete to C_3, the center of B_1 is adjacent to $\text{comp}(B, B_2)$. If B and B_2 are neighbor bags, then the marked edge connecting them can be recomposed. Thus, in this case, $\text{path}(B, B_2)$ contains 3 bags. Let B_4 be the bag in $\text{path}(B, B_2) \setminus \{B, B_2\}$, and b be an unmarked vertex in B_4. It is not difficult to observe that p_1 should be adjacent to b, since p_1 cannot be in C_2. Then $bp_1p_2p_3p_4$ is an induced path and c_1 is adjacent to its end vertices, and thus G contains a small DH obstruction. It is a contradiction. We may assume $B_1 \neq B$.

Similar to the case when B_1 is a complete bag, we can show that the parent of B_1 is B and B is a star whose center is adjacent to B_1. In this case, p_1 is contained in the non-S-attached twin class, as p_1 is not adjacent to p_4. As B has at least 3 vertices, there is a vertex $x \in V(G-S)$ where x is adjacent to p_1, but not adjacent to p_2, p_3, p_4. If x is adjacent to p_4, then we have a small DH obstruction. Otherwise, $xp_1p_2p_3p_4$ is an induced path, and c is adjacent to its end vertices. This contradicts the non-existence of a small DH obstruction.

We conclude that for every induced path $P = p_1p_2p_3p_4p_5$ with $p_3 \in C_2$, there is a bypassing vertex for P and p_3. □

Next, we deal with the case when C_1 is complete to C_2. We prove the case when B is a star bag in Proposition 7.36, and the case when B is a complete bag in Proposition 7.40.

Proposition 7.36. *Suppose C_1 is complete to C_2, and either*

- *$B_2 = B$ and C_2 consists of the center of B_2 or*
- *$B \neq B_2$, and B is a star bag whose center is adjacent to* $\mathrm{comp}(B, B_2)$.

Then for every induced path $P = p_1p_2p_3p_4p_5$ with $p_3 \in C_1$, there is a bypassing vertex for P and p_3.

Proof. For each $i \in \{1, 2\}$ and let $c_i \in C_i$ and $T_i = N_G(C_i) \setminus (C_1 \cup C_2)$. We first observe that there is no child bag B_1' pending to B_1 except the possible child in $\mathrm{path}(B, B_2)$ when $B = B_1$. Suppose there is such a bag, and let D' be the connected component of $D - V(B_1)$ containing B_1'. As D' has no S-attached bags by the choice of B, B_1, B_2, by Lemma 7.19, B_1 is a star whose center is adjacent to D'. Then B_1 becomes a (C_1, C_2)-separator bag, contradicting the assumption that C_1 is complete to C_2. We conclude the claim.

As B_i is the unique S-attached bag in $\mathrm{comp}(B, B_i)$ when $B \neq B_i$, by Lemma 7.19, every bag in $\mathrm{path}(B, B_i) \setminus \{B, B_i\}$ has exactly two neighbor bags. Since either

- $B_2 = B$ and C_2 consists of the center of B_2 or
- B is a star bag whose center is adjacent to $\mathrm{comp}(B, B_2)$,

every neighbor of a vertex in C_1 is contained in $\mathrm{comp}(B, B_1)$ or $\mathrm{comp}(B, B_2)$.

Suppose there is an induced path $P = p_1p_2p_3p_4p_5$ with $p_3 \in C_1$. We will show that there is a bypassing vertex for p_3. If $p_2, p_4 \in S$, then it follows from Lemma 7.14. Without loss of generality, we assume $p_2 \in V(G-S)$. We distinguish cases depending on whether $(T_1 \cap T_2) \cap V(G-S) = \emptyset$ or not.

Case 1. $(T_1 \cap T_2) \cap V(G - S) \neq \emptyset$ **:** We choose a vertex $x \in (T_1 \cap T_2) \cap V(G - S)$. Since every neighbor of a vertex in C_1 is contained in $\text{comp}(B, B_1)$ or $\text{comp}(B, B_2)$, x is contained in $\text{comp}(B, B_1)$ or $\text{comp}(B, B_2)$. Since x is not contained in $C_1 \cup C_2$, x has no neighbors in S.

Claim 7.37. *$(T_1 \cap T_2) \cap S \neq \emptyset$.*

Proof of the Claim. Suppose $T_1 \cap S$ and $T_2 \cap S$ are disjoint. As C_1 is complete to C_2 and (G, S, k) is reduced under Branching Rule 7.2, $T_1 \cap S$ and $T_2 \cap S$ are contained in the same conncected component of $G[S]$. Let P be a shortest path from $T_1 \cap S$ to $T_2 \cap S$ in $G[S]$. Since Pc_2x is an induced path of length at least 3 and c_1 is adjacent to its end vertices, $G[V(P) \cup \{c_1, c_2, x\}]$ contains a DH obstruction, which contradicts the assumption that G is reduced under Branching Rule 7.1. Therefore, $(T_1 \cap T_2) \cap S \neq \emptyset$. $\diamondsuit$

Let $y \in (T_1 \cap T_2) \cap S$. Clearly, x is not adjacent to y.

Claim 7.38. *$((T_1 \setminus T_2) \cup (T_2 \setminus T_1)) \cap S = \emptyset$.*

Proof of the Claim. Suppose there is a vertex u in $((T_1 \setminus T_2) \cup (T_2 \setminus T_1)) \cap S$. Since $xy \notin E(G)$, if $uy \in E(G)$, then $ux \in E(G)$ by (1) of Lemma 7.12. But x has no neighbors in S. Thus, we have $uy \notin E(G)$. Since $\{u, y\} \subseteq T_1 \cap S$ or $\{u, y\} \subseteq T_2 \cap S$, there is an induced path upy for some $p \in S$. We assume $\{u, y\} \subseteq T_1 \cap S$; the symmetric argument holds when $\{u, y\} \subseteq T_2 \cap S$. If $p \in T_1 \cup T_2$, then by (1) of Lemma 7.12, u or p should be adjacent to x, which is a contradiction. On the other hand, by (2) of Lemma 7.12, p cannot be in $S \setminus (T_1 \cup T_2)$. We conclude that $((T_1 \setminus T_2) \cup (T_2 \setminus T_1)) \cap S = \emptyset$. $\diamondsuit$

Suppose that $p_4 \in V(G - S)$. We know that p_2 and p_4 are contained in some bags in $\text{path}(B_1, B_2)$. By symmetry, we assume $|\text{path}(B_1, A_1)| \leq |\text{path}(B_1, A_2)|$, where A_1 and A_2 are bags containing p_2 and p_4, respectively.

Since $p_2p_4 \notin E(G)$, A_1 is not a complete bag, and thus it is a star whose center is adjacent to $\text{comp}(A_1, B_1)$. In case when $P_1 = P_2 = B_2$, we may assume that p_2 is contained in the non-S-attached twin class. Then p_1 should be adjacent to p_4, contradiction.

We may assume that $p_2 \in V(G - S)$ and $p_4 \in (T_1 \cap T_2) \cap S$, because $((T_1 \setminus T_2) \cup (T_2 \setminus T_1)) \cap S = \emptyset$. Since $p_4 \in T_1 \cap T_2$, we have $p_2 \notin C_2$. We can observe that p_2 has no neighbors in S as p_2 is contained in some bag in $\text{path}(B_1, B_2) \setminus \{B\}$, and it is not contained in $C_1 \cup C_2$.

If p_1 is adjacent to c_2, then c_2 is adjacent to the end vertices of an induced path $p_1p_2p_3p_4$, implying that G has a small DH obstruction, which is a contradiction. We may assume that p_1 is not adjacent to c_2. One can observe that in this case, p_1 is in some bag of $\text{path}(B_1, B_2)$, and thus p_1 is adjacent to p_3. This is a contradiction.

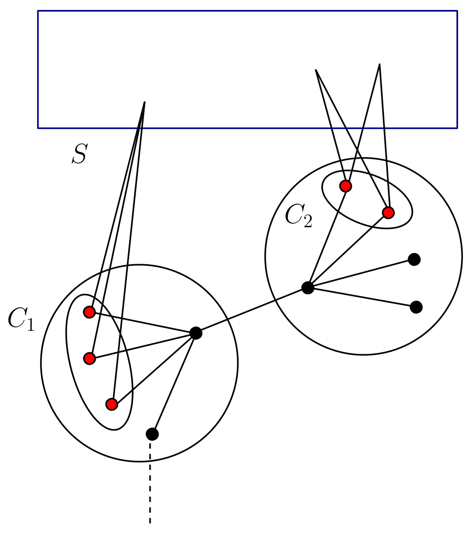


Figure 7.13: The case when $(T_1 \cap T_2) \cap V(G - S) = \emptyset$ in Proposition 7.36.

Case 2. $(T_1 \cap T_2) \cap V(G - S) = \emptyset$ **:** This implies that there are no complete bags in $\text{path}(B_1, B_2) \setminus \{B_1, B_2\}$, and especially, if B_1 or B_2 is a complete bag, then it has no non-S-attached twin class. We first claim that $B = B_1$, $B \neq B_2$ and B is the parent bag of B_2.

Claim 7.39. *$B = B_1$, $B \neq B_2$ and B is the parent bag of B_2.*

Proof of the Claim. Suppose $B \neq B_1$, and let B_1' be the parent bag of B_1. As C_1 is complete to C_2, B_1 is a star whose center is adjacent to B_1' or a complete bag. By Lemma 7.19, there is no child of B_1, and thus B_1 is a leaf bag. Also, by Lemma 7.19, B_1' has exactly two neighbor bags unless $B_1' = B$.

We observe that B_1' should be a star whose center is adjacent to B_1. When B_1 is a star, B_1' is a star whose center is adjacent to B_1, as there is no complete bag in $\text{path}(B_1, B_2) \setminus \{B_1, B_2\}$. When B_1 is a complete bag, if B_1' is a star whose leaf is adjacent to B_1, then we can apply Reduction Rule 7.4 because B_1 contains no non-S-attached twin class. Thus B_1' is a star whose center is adjacent to B_1. Due to Lemma 7.20, such a bag B_1' cannot exist. Therefore, we have $B = B_1$.

Since $B_1 \neq B_2$ by Lemma 7.32, we have $B \neq B_2$. In the same reason, there are no bags in $\text{path}(B, B_2) \setminus \{B, B_2\}$. This implies that B is the parent of B_2. $\Diamond$

See Figure 7.13 for an illustration. Recall that p_2 is contained in $G - S$. Thus, p_2 should be contained in C_2 or B_2 has the non-S-attached twin class and p_2 is contained in this class.

Suppose $p_2 \in C_2$. Then $p_4 \in (T_1 \setminus T_2) \cap S$. If p_4 has a neighbor in T_2, then we have a bypassing vertex. So, we may assume that p_4 has no neighbors in $T_2 \cap S$. As C_1

is complete to C_2 and Branching Rule 7.2 is exhaustively applied, p_4 and $T_2 \cap S$ are contained in the same connected component of $G[S]$. Let P be a shortest path from p_4 to $T_2 \cap S$ in $G[S]$. Then Pp_2 is an induced path of length at least 3 and p_3 is adjacent to its end vertices, and therefore $G[S \cup \{p_2, p_3\}]$ contains a DH obstruction which contradicts the exhaustive application of Branching Rule 7.1.

Now, suppose $p_2 \notin C_2$. This implies that B_2 is a star bag having a non-S-attached twin class, and p_2 is contained in the set. Then p_1 should be a common neighbor of c_2 and p_2. Let P be the shortest path from p_4 to $T_2 \cap S$ in $G[S]$. First assume that p_1 has a neighbor in P. Among neighbors of p_1 in P, we choose the vertex q such that the distance between p_4 and q in P is shortest. Let Q be the subpath of P from p_4 to q. Then p_2p_1Q is an induced path of length at least 3 since p_1 is not adjacent to p_4, and p_3 is adjacent to its end vertices. Therefore, $G[S \cup \{p_1, p_2, p_3\}]$ contains a DH obstruction, contradicting our assumption that G is reduced under Branching Rule 7.1. We may assume that p_1 has no neighbors in P. In this case, $p_2p_1c_2P$ is an induced path of length at least 3, and p_3 is adjacent to its end vertices. Therefore, $G[S \cup \{p_1, p_2, p_3, c_2\}]$ contains a DH obstruction, contradicting our assumption that G is reduced under Branching Rule 7.1.

We conclude that for every induced path $P = p_1p_2p_3p_4p_5$ with $p_3 \in C_2$, there is a bypassing vertex for P and p_3. □

Proposition 7.40. *Suppose C_1 is complete to C_2, $B \neq B_1$, and B is a complete bag. Then B_1 contains a non-S-attached twin class C_1' and for every induced path $P = p_1p_2p_3p_4p_5$ with $p_3 \in C_1'$, there is a bypassing vertex for P and p_3.*

Proof. For each $i \in \{1, 2\}$ and let $c_i \in C_i$ and $T_i = N_G(C_i)$. Let B_3 be the parent bag of B_1. As C_1 is complete to C_2, B_1 is either a complete bag or a star whose center is adjacent to B_3. We observe that B_1 has a non-S-attached class, and $(T_2 \setminus T_1) \cap V(G - S) \neq \emptyset$.

Claim 7.41. *B_1 contains a non-S-attached class.*

Proof of the Claim. Suppose for contradiction that B_1 has no non-S-attached class, that is, its unmarked vertices form one S-attached twin class. We verify that there is no child bag of B_1. Suppose for contradiction that there is a child B_1' of B_1 and let D' be the component $\text{comp}(B_1, B_1')$. Since D' contains no S-attached bag, by Lemma 7.19, B_1 should be a star whose center is adjacent to B_1', which is a contradiction. Also, every bag in $\text{path}(B, B_1) \setminus \{B, B_1\}$ is not a (B, B_1)-separator bag, and by Lemma 7.19, it has exactly two neighbor bags.

Assume B_1 is a complete bag. Then its parent B_3 is a star and thus $B \neq B_3$. By Lemma 7.20, the center of B_3 is not adjacent to B_1. Thus, the center of B_3 is adjacent to its parent. As B_1 has no non-S-attached class, we can apply Reduction Rule 7.4 to reduce the split decomposition, which is a contradiction. Assume B_1 is a star whose center is adjacent to its parent. Similarly, by Lemma 7.20, B_3 is not a star whose center is adjacent to B_1. Thus, we may assume the parent of B_1 is a complete bag, but in this case, we can apply Reduction Rule 7.4. ◊

Let C_1' be the non-S-attached twin class in B_1. As C_1 and C_1' have the same neighborhood in $G - S$, C_1' is complete to C_2.

Claim 7.42. *$(T_2 \setminus T_1) \cap V(G - S)$ contains a vertex that has no neighbors in S.*

> *Proof of the Claim.* If B_1 is a star bag, then $C_1' \subseteq (T_2 \setminus T_1) \cap V(G - S)$ and $y \in C_1'$. If B_1 is a complete bag, then since B is a complete bag, $B_3 \neq B$ and the unmarked vertices in B_3 are contained in $(T_2 \setminus T_1) \cap V(G - S)$. Let y be an unmarked vertex in B_3. By the choice of y, y has no neighbors in S. $\Diamond$

Let y be a vertex in $(T_2 \setminus T_1) \cap V(G - S)$ having no neighbors in S.

Suppose there is an induced path $P = p_1p_2p_3p_4p_5$ with $p_3 \in C_1'$. We will prove that there is a bypassing vertex for P and p_3. Let D' be the connected component of $D - V(B)$ containing the parent of B.

Claim 7.43. *p_2 and p_4 are contained in $(T_1 \cap T_2) \cap V(G - S)$.*

> *Proof of the Claim.* Note that p_2 or p_4 is contained in either D' or path(B_1, B_2). If both p_2 and p_4 are contained in D', then this is clear. If both p_2 and p_4 are contained in path(B_1, B_2), then without loss of generality, we assume that $|\text{path}(B_1, A_1)| \leq |\text{path}(B_1, A_2)|$ where A_1 and A_2 are bags containing p_2 and p_4, respectively. Since there is no (B_1, B_2)-separator bag, p_1 should be adjacent to p_4, a contradiction. Lastly, we assume that one of p_2 and p_4 is in D', but the other is in path(B_1, B_2). By symmetry we assume $p_2 \in V(D')$. If $p_4 \in \text{path}(B_1, B)$, then p_4 is contained in a complete bag, and thus p_2 is adjacent to p_4. If $p_4 \in \text{path}(B, B_2)$, then p_4 is clearly adjacent to p_2, as B is a complete bag. Both cases are not possible. $\Diamond$

Suppose $(T_1 \cap T_2) \cap S \neq \emptyset$. Let $x \in (T_1 \cap T_2) \cap S$. Since we know that p_3 has no neighbors in S, we have $p_3x \notin E(G)$, and by (1) of Lemma 7.12, x should be adjacent to both p_2 and p_4. Thus, x is a bypassing vertex, as required. We may assume that $(T_1 \cap T_2) \cap S = \emptyset$.

Since C_1 is complete to C_2, by Branching Rule 7.2, we know that $T_1 \cap S$ and $T_2 \cap S$ are contained in the same connected component of $G[S]$. Let P be a shortest path from $T_1 \cap S$ to $T_2 \cap S$. If P has length at least 2, then $G[V(P) \cup \{c_1, c_2\}]$ is an induced cycle of length at least 5, contradicting our assumption that G is reduced under Branching Rule 7.1. Thus, P has length 1. Let q_1q_2 be the path where q_i is a neighbor of c_i for each $i \in \{1, 2\}$.

Note that $q_1 \in T_1 \setminus T_2$, $q_2 \in T_2 \setminus T_1$ and $p_2, p_4 \in T_1 \cap T_2$. If q_1 or q_2 is adjacent to one of p_2 and p_4, the by (1) of Lemma 7.12, it is adjacent to both p_2 and p_4. This means that it becomes a bypassing vertex, as required. Therefore, we may assume that for each $i \in \{1, 2\}$, q_i is adjacent to neither p_2 nor p_4. So, $p_2c_1q_1q_2$ is an induced path of length 3, and c_2 is adjacent to its end vertices. This implies that G has a small DH obstruction, which is a contradiction.

We conclude that for every induced path $P = p_1p_2p_3p_4p_5$ with $p_3 \in C_2$, there is a bypassing vertex for P and p_3. □

Proposition 7.44. *If D is a reduced canonical split decomposition of a connected component of $G - S$, then D is empty.*

Proof. Suppose a reduced canonical split decomposition D of a connected component of $G - S$ contains a vertex. If it contains at most one S-attached twin class, then Reduction Rule 7.1 can be applied. We may assume that D contains at least two S-attached twin classes.

We choose a root bag, and let B be a farthest bag from the root bag such that there are two descendant bags B_1 and B_2 of B having distinct S-attached twin classes C_1 and C_2, respectively. By Lemma 7.32, $B_1 \neq B_2$, and one of the following happens:

(1) The distance from C_1 to C_2 in $G - S$ is 2 and the unique (C_1, C_2)-separator bag is contained in $\text{comp}(B, B_i)$ for some $i \in \{1, 2\}$.

(2) C_1 is complete to C_2 and either

- B is a star bag and C_i is the set consisting of the center of B for some $i \in \{1, 2\}$, or
- B is a star bag whose center is adjacent to $\text{comp}(B, B_i)$ for some $i \in \{1, 2\}$.

(3) C_1 is complete to C_2 and B is a complete bag.

If (1) happens, then by Proposition 7.33, Reduction Rule 7.3 is applied to remove C_i. If (2) happens, then by Proposition 7.36, Reduction Rule 7.3 is applied to remove C_{3-i}. If (3) happens, then by Proposition 7.40, Reduction Rule 7.3 is applied to remove the non-S-attached twin class in one of B_1 and B_2. But this contradicts the assumption that D is reduced. □

7.5 The Algorithm, Lower Bounds and Applications

Our goal in this section is to give a proof of our main result, Theorem 7.46, and obtain corresponding lower bounds.

7.5.1 The Algorithm

Below, we use the presented reduction and branching rules to give an algorithm for Disjoint Distance-Hereditary Vertex Deletion. This is then followed by a proof of our main algorithmic result.

Theorem 7.45. Disjoint Distance-Hereditary Vertex Deletion *can be solved in time* $\mathcal{O}(6^{k+\mathrm{cc}(G[S])} \cdot |V(G)|^6(|V(G)| + E(G)))$*.*

Proof. Let (G, S, k) be an instance of Disjoint Distance-Hereditary Vertex Deletion. We exhaustively apply Branching Rules 7.1–7.2 and Reduction Rules 7.1–7.5. We prove that one of rules can be applied until $G - S$ is empty or k becomes 0. In both cases, we can test whether the resulting instance is distance-hereditary or not in polynomial time, and output an answer. Suppose k does not reach 0. Then by Proposition 7.44, $G - S$ contains no vertices. Therefore, the resulting instance satisfies that $G - S$ is empty, as mentioned.

We argue that the runtime bounds hold. For convenience, we will denote $|V(G)|$ by n and $|E(G)|$ by m. First notice that each branching rule reduces either k or the number of connected components in $G[S]$ and branches into at most 6 subinstances. Moreover, none of the reduction rules change k or the number of components in S. Hence a branching rule is applied at most $k + \mathrm{cc}(G[S])$ times. Similarly, every reduction rule reduces either the number of vertices in $G - S$ or the number of bags in canonical split decomposition of $G - S$. Therefore, it is not hard to observe that the branching tree of the algorithm will have at most $6^{k+\mathrm{cc}(G[S])}$ leaves and each leaf will be in depth at most $\mathcal{O}(n)$ and hence the branching tree will have at most $\mathcal{O}(6^{k+\mathrm{cc}(G[S])} \cdot n)$ nodes. In the following we will discuss that the runtime in every node will not exceed $\mathcal{O}(n^5(n + m))$. In each node, we go through the branching and reduction rules, in the order they are introduced, and apply the first rule that can be applied. Let us start with detecting and applying Branching Rule 7.1. Our algorithm is going through all sets $X \subseteq G - S$ of size at most 5 and checking, whether $G[S \cup X]$ is distance-hereditary. It follows from Theorems 7.7 and 7.8 that we check whether a graph is distance-hereditary in time $\mathcal{O}(n + m)$. If the graph is not distance-hereditary, application of the rule can be done in constant time. Hence, the Branching Rule 7.1 can be verified in time $\mathcal{O}(n^5(n + m))$. Similarly, for Branching Rule 7.2 for every set $X \subseteq V(G - S)$ of size at most 5, we can in time $\mathcal{O}(n + m)$, e.g. using breadth-first search, verify that the neighborhood of X is in the same connected component and the same running time bound follows. After verifying that the graph is actually reduced under Branching Rules 7.1 and 7.2 it follows from Proposition 7.16 that we can in time $\mathcal{O}(n^6)$ either apply one of Reduction Rules 7.1–7.5 or correctly deduce that the graph is reduced also under Reduction Rules 7.1–7.5. Hence, the whole algorithm for Disjoint Distance-Hereditary Vertex Deletion can be implemented in time $\mathcal{O}(6^{k+\mathrm{cc}(G[S])} \cdot n^6(n + m))$. □

Theorem 7.46. Distance-Hereditary Vertex Deletion *can be solved in time* $\mathcal{O}(37^k \cdot |V(G)|^7(|V(G)| + |E(G)|))$*.*

Proof. We apply the standard iterative compressing technique. The algorithm involves a two-step reduction of Distance-Hereditary Vertex Deletion: we first reduce Distance-Hereditary Vertex Deletion to the Compression problem, which reduces to Disjoint Distance-Hereditary Vertex Deletion.

For convenience, we denote for this proof $|V(G)| = n$ and $|E(G)| = m$. Fix an arbitrary labeling $v_1, \dots, v_n$ of $V(G)$ and let G_i be a the graph $G[\{v_1, \dots, v_i\}]$ for $1 \leq i \leq n$. From $i = 1$ up to n, we consider the following the Compression problem for Distance-Hereditary Vertex Deletion: given a graph G_i and $S_i \subseteq V(G_i)$ such that $G_i - S_i$ is distance-hereditary and $|S_i| \leq k+1$, we aim to find a set $S_i' \subseteq V(G_i)$ such that $G_i - S_i'$ is distance-hereditary and $|S_i'| \leq k$, if one exists, and output No otherwise. Since distance-hereditary graphs are closed under taking induced subgraphs, (G,k) is Yes-instance of Distance-Hereditary Vertex Deletion if and only if (G_i, S_i) is a Yes-instance for Compression for all i, where (G_i, S_i) is a legitimate instance. Hence we correctly output that (G,k) is No-instance of Distance-Hereditary Vertex Deletion if (G_i, S_i) is a No-instance for some i. Moreover, if S_i' is a solution to the i-th instance of Compression, then $(G_{i+1}, S_i' \cup \{v_{i+1}\})$ is a legitimate instance for $(i+1)$-th instance of Compression.

Given an instance (G,S) of Compression, we enumerate all possible intersections I of S and a desired solution to (G,S). For each guessed set I, we solve the instance $(G - I, S \setminus I, k - |I|)$ of Disjoint Distance-Hereditary Vertex Deletion using Theorem 7.45. Note that (G,S) is Yes-instance if and only if $(G - I, S \setminus I, k - |I|)$ is Yes-instance for some $I \subseteq S$. If S' is a solution to $(G - I, S \setminus I, k - |I|)$, then clearly $S' \cup I$ is a solution to the instance (G,S) of Compression. Conversely, if S' is a solution to the instance (G,S) of Compression then for the set $I = S \cap S'$ the instance $(G - I, S \setminus I, k - |I|)$ is Yes-instance for Disjoint Distance-Hereditary Vertex Deletion. Therefore, using the algorithm from Theorem 7.45 we can correctly solve Distance-Hereditary Vertex Deletion.

It remains to prove the complexity of the algorithm. Given an instance (G,S) we guess at most $\binom{k+1}{i}$ sets $I \subseteq S$ of size i for each $1 \leq i \leq k$. Note that $S \setminus I$ has size at most $k+1-i$, and in particular $G[S]$ has at most $k+1-i$ connected components. Therefore, we can solve the resulting instance $(G - I, S \setminus I, k - i)$ of Disjoint Distance-Hereditary Vertex Deletion in time $\mathcal{O}(6^{2k-2i+1} \cdot n^6(n+m)) = \mathcal{O}(36^{k-i} \cdot n^6(n+m))$. Summing up, Distance-Hereditary Vertex Deletion can be solved by running an algorithm for Compression at most n times, which yields the claimed running time

$$n \cdot \sum_{i=0}^{k} \binom{k+1}{i} \cdot \mathcal{O}(36^{k-i} \cdot n^6(n+m)) = \mathcal{O}(37^k \cdot n^7(n+m)).$$

Note that the equality follows from the use of the binomial theorem, which states that $\sum_{i=0}^{n} \binom{n}{i} a^i b^{n-i} = (a+b)^n$ (see, e.g., Chapter 10 in Cygan et al. [53]). □

7.5.2 Lower Bounds

Here we present our tight lower bound result based on the fact that the classical Vertex Cover problem cannot be solved in subexponential time unless ETH fails (recall Theorem 2.5).

Theorem 7.47. *There is no* $2^{o(k)} \cdot |V(G)|^{\mathcal{O}(1)}$ *algorithm for* Distance-Hereditary Vertex Deletion *unless ETH fails.*

Proof. For a graph G, we will denote $|V(G)|$ by n and $|E(G)|$ by m. For contradiction suppose there exists an algorithm for solving the Distance-Hereditary Vertex Deletion problem in time $2^{o(k)} \cdot n^{\mathcal{O}(1)}$. We show that we can solve Vertex Cover in time $2^{o(k)} \cdot n^{\mathcal{O}(1)}$. Let (G, k) be an instance of Vertex Cover problem. We construct a graph G' as follows. We replace every edge uv of G with two vertex disjoint paths of length 3 between u and v. Note that for every edge uv in G the two disjoint paths of length 3 in G' form an induced subgraph isomorphic to C_6. Moreover we have $|V(G')| = |V(G)| + 4|E(G)|$. We claim that G has a vertex set S of size at most k such that $G - S$ has no edges if and only if G' has a vertex deletion set of size at most k to a distance-hereditary graph. Suppose that G has such a vertex cover S. It is easy to confirm that $G' - S$ is a disjoint union of subdivisions of stars, which is distance-hereditary.

For the converse direction, suppose G' has a distance-hereditary vertex deletion set S of size at most k. Let us fix an arbitrary edge uv in G. Note that no DH obstruction contains a pendant vertex. Hence we observe that if H is a DH obstruction containing a vertex t on a shortest $u - v$ path in G', then H contains both vertices u and v as well. Therefore, if $t \in S$, then also graphs $G' - (S \setminus \{t\} \cup \{u\})$ and $G' - (S \setminus \{t\} \cup \{v\})$ are distance-hereditary. Since the choice of the edge uv was arbitrary, we can find a set T, such that $T \subseteq V(G)$, $|T| \leq |S|$, and $G' - T$ is a distance-hereditary graph. Clearly for every edge uv in G, T contains u or v, otherwise $G' - T$ contains an induced C_6. We conclude that T is a vertex cover of G. □

7.5.3 Example Applications

There is an established line of research studying the algorithmic applications of vertex deletion sets to specific graph classes [92, 79]. In this context, it is natural to ask whether Theorem 7.46 allows the development of single-exponential algorithms for problems parameterized by the size of a vertex deletion set to distance-hereditary graphs.

Clearly, any problem that is FPT when parameterized by clique-width (and rank-width) must also be FPT when parameterized by the size of a vertex deletion set to distance-hereditary graphs. However, the existence of a single-exponential FPT algorithm parameterized by clique-width does not immediately imply that the problem also admits a single-exponential FPT algorithm parameterized by our parameter, since the addition of k vertices to a graph may increase clique-width by a factor of up to 2^k [111]. On the other hand, known FPT algorithms parameterized by rank-width usually do not have a single-exponential dependency on the parameter. As a consequence, one cannot obtain the following examples of single-exponential algorithms by simply solving these problems via known FPT algorithms parameterized by rank-width or clique-width.

Lemma 7.48. Vertex Cover *and* 3-Coloring *admit a single-exponential FPT algorithm when parameteried by the size of a vertex deletion set to distance-hereditary graphs.*

Proof. For each of the presented problems, we always begin by invoking Theorem 7.46 to compute a vertex deletion set X to distance-hereditary graphs of size at most k.

In the case of Vertex Cover, we can apply standard branching algorithms to solve the problem. In particular, we begin by branching over the at most 2^k options of how X intersects with a (hypothetical) solution; let X_1 be one such subset of X and let $X_2 = X \setminus X_1$. After branching we proceed by testing the validity of a branch (i.e., whether each edge with both endpoints in X is covered by X_1). For each valid branch, we delete X and the set Z of all neighbors of X_2 in $G - X$. Next, we find a minimum vertex cover C in the remaining distance-hereditary subgraph of G in polynomial time. Finally, for each branch we compare the desired solution size with $|C \cup X_1 \cup Z|$; clearly, a graph is a YES-instance of Vertex Cover if and only if at least one selection of X_1 results in a value of $|C \cup X_1 \cup Z|$ which is at most the desired solution size.

For 3-Coloring, we also begin by branching over the at most 3^k 3-colorings of X. For each such proper 3-coloring of X, we construct an instance of 3-List Coloring as follows: the input graph is $G - X$, and the list of admissible colors for each vertex v contains all colors that are not used by a neighbor of v in X. The 3-List Coloring problem can be solved in polynomial time on distance-hereditary graphs: indeed, the problem can easily be reduced to the MSO_1 model checking problem over labeled graphs with (at most) 8 labels. Since $G - X$ has rank-width at most 1, the polynomial-time tractability of the problem follows for instance from Courcelle's Theorem [50]. All that remains now is to test whether at least one of the considered 3^k branches gives rise to a yes-instance of 3-List Coloring on $G - X$. □

7.6 Summary and Open Questions

We conclude with a few remarks on why we believe that the presented algorithm is of high interest. First, it intrinsically exploits the properties guaranteed by distinct, seemingly unrelated characterizations of distance-hereditary graphs; this approach can likely be used to design or improve algorithms for other vertex deletion problems. Second, it uses highly nontrivial reduction rules which simplify canonical split decompositions, and an adaptation or extension of the presented rules could be highly relevant for other graph classes characterized by special canonical split decompositions, such as parity graphs [43] or circle graphs [91]. Third, it is the first of its kind which targets a "full" class of graphs of bounded rank-width (contrasting previous results for specific subclasses of graphs of rank-width 1 [122, 5, 136, 133]).

It is worth noting that there remains a number of interesting open problems in this general area. Perhaps the most prominent one is the question of whether vertex deletion to graphs of rank-width c, for any constant c, admits a single-exponential FPT algorithm.

Our algorithm represents the first steps in this general direction. Recently, Kim and Kwon [137] gave a polynomial kernel for Distance-Hereditary Vertex Deletion. The existence of a polynomial kernel or an approximation algorithm for such vertex deletion problems for $c > 1$ remains open.

Notes

The results in this chapter appeared in a conference paper in the proceedings of 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2016) [71].

CHAPTER 8

Towards a Polynomial Kernel for Directed Feedback Vertex Set

The existence of a polynomial kernel for DFVS is a challenging open problem in the area of parameterized algorithms. In particular, since the breakthrough result of Chen, Liu, Lu, O'Sullivan and Razgon [42] which showed that DFVS is FPT, virtually no progress has been made towards settling the existence of a polynomial kernel for this problem. Given the apparent difficulty of this question, it makes sense to investigate more restrictive parameterizations, which may make it easier to obtain a kernel for DFVS. Here, we investigate the existence of a polynomial kernel for DFVS under natural parameters which upper-bound the size of the directed feedback vertex set; in particular, we consider the size of a feedback vertex set of the underlying undirected graph of the DFVS instance.

Results

Before, we get to the results, we point out that in this chapter we actually consider undirected and directed multigraphs, since they are more suitable for dealing with reduction rules for DFVS. Therefore in the course of this chapter we will refer to multigraphs simply as graphs. Furthermore, throughout this chapter, for a directed graph D we denote by $\overline{D}$ the undirected multigraph obtained from D after replacing every arc $(u, v) \in E(D)$ with an edge $\{u, v\}$; note that if there are arcs in both directions between u and v, in this case we also say that there is a bidirectional arc between u and v, then $\overline{D}$ will contain two edges between u and v. We call $\overline{D}$ the underlying undirected graph of D. The problem we are interested in can be formally stated as follows.

Directed Feedback Vertex Set parameterized by FVS (DFVS[FVS])

Instance: A digraph D, an integer p, and a set F such that F is an UFVS of D.
Parameter: $|F|$
Question: Does there exist a vertex subset of size at most p that intersects every cycle in D?

Since every UFVS (undirected feedback vertex set) of D is also a DFVS (directed feedback vertex set) of D, we may assume without loss of generality that $p \leq |F|$ for every instance of DFVS[FVS]. Furthermore, we use k to denote the size of F. Our first result is a polynomial kernel for DFVS[FVS], formally stated below. We note that this may be viewed as an intermediate step towards resolving the existence of a polynomial kernel for DFVS.

Theorem 8.1. *There is a kernel with $\mathcal{O}(k^4)$ vertices for* DFVS[FVS].

Interestingly, the existence of a polynomial kernel for DFVS parameterized by the solution size remains open even in the restricted setting of planar graphs. While our Theorem 8.1 naturally also provides a polynomial kernel for DFVS[FVS] on planar graphs, as our second main contribution we show that one can in fact obtain a significantly stronger result not only on planar graphs, but on all graphs embeddable on orientable surfaces.

Theorem 8.2. *There is a kernel with $\mathcal{O}(k)$ vertices for* DFVS[FVS] *when the input digraph is embeddable on a surface of constant genus.*

Organization of the Chapter

First, in Section 8.1, we deal with the polynomial kernel for the case of the general directed graphs. Afterwards, in Section 8.2 we give a linear kernel for digraphs that can be embedded in a given fixed surface. In both section, in order to describe our kernelization algorithm, we present a series of reduction rules. We prove the soundness of each reduction rule immediately after presenting its description, unless the soundness is obvious. Moreover, the reduction rules we present will be executed in the order in which they appear. That is, if at any point we may apply Reduction Rule i as well as Reduction Rule j where $i < j$, we will execute Reduction Rule i.

8.1 A Polynomial Kernel for DFVS[FVS]

Note that every FVS of $\overline{D}$ is also a DFVS of D. Given an instance D of DFVS, our kernelization algorithm for DFVS parameterized by FVS first computes a 2-approximate FVS S of $\overline{D}$ (using for instance the algorithm given in [16]) and then uses S to reduce the instance in polynomial-time into an equivalent instance with at most $\mathcal{O}(|S|^4)$ vertices.

Hence in the following we will assume that D is a directed graph and S is a FVS of $\overline{D}$ (and hence also a DFVS of D) of size k. Our first two reduction rules are sound because (a) neither sinks nor sources can appear on a directed cycle and (b) if a vertex v has exactly one in-neighbor u in D then every directed cycle containing v has to use the arc (u, v) (a symmetric statement holds for vertices with exactly one out-neighbor).

Reduction Rule 8.1. Delete all sources and sinks from D.

Reduction Rule 8.2. Let l be an arbitrary vertex in D.

- If l has exactly one out-neighbor $p \in V(D)$, then we contract the arc (l, p) into a new vertex l^*.
- If l has exactly one in-neighbor $p \in V(D)$, then we contract the arc (p, l) into a new vertex l^*.

After the exhaustive application of these two rules, we may assume, w.l.o.g , that the digraph D has no sinks or sources, and furthermore that every vertex has at least 2 in-neighbors and at least 2 out-neighbors. We now state one of our main reduction rules.

Reduction Rule 8.3. Let u and v be two (not necessarily distinct) vertices in S such that there are at least $k + 1$ internally vertex-disjoint directed u-v paths in D. Then,

- if $u \neq v$, we add an arc from u to v to D, or
- if $u = v$, we remove u from D and decrease the parameter k by one.

Proof of soundness. Let $u, v \in S$ be as above and let D' be the digraph obtained from D after applying the reduction rule. If $u = v$ then clearly every DFVS for D of size at most k contains u, which shows the soundness of the reduction rule.

If on the other hand $u \neq v$, we will show that a set $S' \subseteq V(D) = V(D')$ of size at most k is a DFVS for D if and only if it is also a DFVS for D'. The backward direction is trivial because D is a subgraph of D'. For the forward direction let S' be a DFVS for D of size at most k and assume for a contradiction that S' is not a DFVS for D'. Then $D' \setminus S'$ contains a directed cycle C that contains the arc (u, v). Hence $D' \setminus S'$ and thus also $D \setminus S'$ contains a directed v-u path P. Moreover, since S' has size at most k and there are at least $k + 1$ vertex-disjoint directed u-v paths in D, we conclude that there is a u-v path P' in $D \setminus S'$. But then $P \cup P'$ must contain a directed cycle, which is also a directed cycle in $D \setminus S'$, a contradiction to our assumption that S' is a DFVS of D. □

We will use Reduction Rule 8.3 to reduce the number of vertices in $D \setminus S$ that 'directly contribute' to (pairs of) vertices in S. The following definition will allow us to formalize this idea.

Definition 8.3. Let (u, v) be an ordered pair of vertices in S. If $u \neq v$, then we refer to (u, v) as a potential arc in $D[S]$ and if additionally $(u, v) \notin D$ then we refer to (u, v) as a non-arc. If on the other hand $u = v$, then we refer to (u, v) as a self-loop. We say that a vertex $v \in V(D) \setminus S$ contributes to a potential arc or self-loop (u, w) , if $(u, v) \in E(D)$ and $(v, w) \in E(D)$.

After the exhaustive application of Reduction Rule 8.3, we have the following structural observation regarding the input.

Observation 8.4. *For every $u \in S$ there are at most k internally vertex-disjoint u-u paths in D; moreover for every two distinct vertices u and v in S with $(u, v) \notin E(D)$, there are at most k vertex disjoint u-v paths in D. As a result, for every non-arc or self-loop (u, v), there are at most k vertices that contribute to (u, v).*

Since S has at most k vertices and at most $k(k-1)$ ordered pairs of vertices, Observation 8.4 implies the following.

Observation 8.5. *There are at most $k^2(k-1)$ vertices in $D \setminus S$ that contribute to some non-arc of $D[S]$. Moreover, there are at most k^2 vertices in $D \setminus S$ that contribute to some self-loop of $D[S]$.*

Our next aim is to bound the number of vertices in $A = D \setminus S$ in terms of k. Towards achieving this we will distinguish these vertices in terms of size of their neighborhood in $D \setminus S$. We therefore denote by A_0, A_1, A_2, and $A_{\geq 3}$ the sets of all vertices in A that have 0, 1, 2, and at least 3 neighbors, respectively, in $D \setminus S$.

8.1.1 Bounding A_0, A_1 and $A_{\geq 3}$.

Note that Observation 8.5 already provides a bound for the number of vertices in A_0 that contribute to some self-loop of $D[S]$. Hence, in order to bound A_0, it is sufficient to provide a bound for the remaining vertices, in the following denoted by A_0', in A_0. In the following let v be a vertex in A_0'. Because of Rule 8.1 v must have at least one in-neighbor and one out-neighbor in S. Consequently, v contributes to at least one potential arc of $D[S]$.

Reduction Rule 8.4. If v does not contribute to a non-arc of $D[S]$, then we remove v from D.

Proof of soundness. Let v be as above and let D' be the directed graph obtained from D after deleting v. We show that a set $S' \subseteq V(D)$ is a DFVS for D if and only if S' is a DFVS for D'. The forward direction of this claim is trivial because D' is a subgraph of D. Towards showing the backward direction let S' be a DFVS for D' and assume for a contradiction that S' is not a DFVS for D. Then there must exist a directed cycle C in $D \setminus S'$ that contains v as well as two arcs (s, v) and (v, s') for some $s, s' \in S$. Because

v does not contribute to a self-loop of $D[S]$, we have that $s \neq s'$. Because v does not contribute to a non-arc of $D[S]$, it follows that $(s, s') \in E(D)$. Hence the arc (s, s') together with the directed path from s' to s contained in C forms a directed cycle in $D' \setminus S'$, a contradiction to our assumption that S' is a DFVS for D'. □

After the exhaustive application of the above rule, we obtain that v contributes to some non-arc of $D[S]$ and hence together with Observation 8.5, we obtain the following.

Observation 8.6. *There are at most $k^2(k-1)$ vertices in A_0'.*

Bounding A_1. We now present the reduction rules we use to bound the size of A_1, i.e., the number of leaves in A. Again it is sufficient to bound the number of vertices in A_1 that do not contribute to some self-loop of $D[S]$, in the following denoted by A_1'. Namely, we will introduce a reduction rule that ensure that every vertex in A_1' contributes to at least one non-arc in $D[S]$. Together with Observation 8.5 this then bounds the size of A_1'. Recall that at this point every vertex in D has at least two in-neighbors and at least two out-neighbors and since moreover every vertex in A_1' does not contribute to a self-loop, we obtain that every vertex in A_1' has at least one in-neighbor and at least one out-neighbor in S that are distinct. Hence we obtain:

Observation 8.7. *Every vertex in A_1' has at least one in-neighbor and at least one out-neighbor in S and hence every vertex in A_1' contributes to a potential arc of $D[S]$.*

The next reduction rule reduces leaves that do not contribute to a non-arc of $D[S]$.

Reduction Rule 8.5. Let $l \in A_1'$ and let p be the unique neighbor of l in $D \setminus S$. If l does not contribute to a non-arc of $D[S]$, then:

- if $(l, p) \in E(D)$, then we delete all arcs from l to vertices in S,
- if $(p, l) \in E(D)$, then we delete all arcs from vertices in S to l.

Proof of soundness. We only show the soundness of the first part since the proof for the soundness of the second part is analogous. Let D' be the directed graph obtained from D after deleting all arcs from l to vertices in S. We will show that any set $S' \subseteq V(D) = V(D')$ is a DFVS of D if and only if S' is a DFVS of D'. The forward direction of this claim is trivial because D' is a subgraph of D. Towards showing the backward direction let S' be a DFVS for D' and assume for a contradiction that S' is not a DFVS for D. Then there is a cycle C in $D \setminus S'$ that contains exactly one of the deleted arcs, say the arc (l, s') with $s' \in S$, from l to some vertex in S. Because the only incoming arcs of l in D are arcs from vertices in S, the cycle C must also contain exactly one arc from some vertex say $s \in S$ to l. Because l does not contribute to any non-arc of $D[S]$, we conclude that $(s, s') \in E(D)$. But then the directed path from s' to s contained in C together with the arc (s, s') forms a directed cycle in $D' \setminus S'$, a contradiction to our assumption that S' is a DFVS for D'. □

Note that after application of Rule 8.5, l will only have either in-neighbors or out-neighbors in S and can hence be reduced further using Rule 8.2. Consequently, after the exhaustive application of the above rules, we conclude that every vertex in A'_1 contributes to at least one non-arc of $D[S]$. Due to Observation 8.5 we conclude that there are at most $k^2(k-1)$ vertices in A'_1. Finally, since $\overline{D \setminus S}$ is a forest and the number of vertices of degree at least 3 in a forest is at most equal to the number of leaves minus two, we get the following.

Observation 8.8. *There are at most $k^3 - 2$ vertices in $A_{\geq 3}$.*

Note that at this point, we have bounded the size of the sets A_0, A_1 and $A_{\geq 3}$ and the only set that remains is A_2.

8.1.2 Bounding A_2

Our next aim is to bound the number of vertices in A_2.

Definition 8.9. Let v be a vertex in A_2. We say that v is a sink-vertex or a source-vertex if the two arcs of $D \setminus S$ incident on it are both incoming arcs or outgoing arcs, respectively. Otherwise we say that v is a balanced-vertex.

Note that due to Reduction Rule 8.2, there are no balanced vertices in $D \setminus S$ which have *no* neighbors in S. This is because otherwise, we would have already contracted one of the two arcs incident to v in $D \setminus S$. Therefore, at this point, we infer the following.

Observation 8.10. *Every vertex in A_2 has at least one neighbor in S.*

Definition 8.11. Let $P = (v_1, \ldots, v_r)$ be a directed path of maximum length in $D \setminus S$ whose internal vertices are in A_2. Then we say that P is a *path segment in $D \setminus S$*. We say that P is an *outer path segment* if at least one of its endpoints is not in A_2, otherwise we say that P is an *inner path segment*.

Note that path segments are by definition *directed paths*. Our strategy now is to obtain a bound on the total number of path segments and then proceed to bound the length of each path segment.

We first bound the number of outer path segments in $D \setminus S$ as follows. Let G be the undirected graph obtained from $\overline{D} \setminus S$ after contracting all edges which are incident to at least one vertex of degree 2. Then the number of outer path segments in $D \setminus S$ is equal to two times the number of edges of G. Because G is a forest without degree two vertices it holds that the number of edges of G is equal to the number of leaves plus the number of non-leaves in G minus one. Hence the number of outer path segments is at most $2(|A'_1 \cup A_{\geq 3}| - 1)$, which together with the already obtained bound on these sets and Observation 8.8 allows us to infer the following.

Observation 8.12. *The number of outer path segments in $D \setminus S$ is at most $4k^2(k-1)+2k^2-6$.*

In order to bound the number of inner path segments, we need to introduce a new reduction rule. We begin by defining the notion of a path segment 'contributing' to a potential arc.

Definition 8.13. We say that a path segment $P = (v_1, \ldots, v_r)$ *contributes* to a potential arc (s, s') of $D[S]$ if there are i and j with $1 \leq i \leq j \leq r$ such that $(s, v_i) \in E(D)$ and $(v_j, s') \in E(D)$. Moreover, we say that P contributes to a self-loop if there are i and j with $1 \leq i \leq j \leq r$ such that $(s, v_i) \in E(D)$ and $(v_j, s) \in E(D)$ for some $s \in S$.

Reduction Rule 8.6. If an inner path segment does not contribute to a non-arc or to a self-loop of $D[S]$, then we remove all internal vertices of P.

Proof of soundness. Let $P = (v_1, \ldots, v_r)$ and let D' be the directed graph obtained from D after deleting all internal vertices of P. We claim that a set S' of size at most k is a DFVS for D if and only if it is a DFVS for D'. The forward direction is trivial since D' is a subgraph of D. For the backward direction let S' be a DFVS for D' of size at most k and suppose for a contradiction that S' is not a DFVS for D. Then there is a directed cycle C in $D \setminus S'$ that contains at least one internal vertex of P. Moreover, because P is an inner path segment (and hence both of its endpoints are either sink- or source-vertices) the path P can only be entered and left by the cycle C via vertices in S. Hence C contains at least one directed subpath that enters P from some vertex say $s \in S$ and leaves P through some vertex say $s' \in S$. Because P does not contribute to a self-loop, we infer that $s \neq s'$ for every such directed subpath of P. Furthermore, because P does not contribute to a non-arc of $D[S]$, we have that $(s, s') \in E(D)$ for every such directed subpath of C. It follows that we can replace all directed s-s' subpaths in C with $s, s' \in S$ and all internal vertices from P, with the arc (s, s') which we know by our assumption, exists in $D[S]$ and thereby obtain a cycle C' in $D' \setminus S'$, a contradiction to our assumption that S' is a DFVS for D'. $\square$

After the exhaustive application of the above rule, we obtain:

Observation 8.14. *Every inner path segment contributes to at least one non-arc or self-loop of $D[S]$.*

Because every pair of inner path segments that contribute to some non-arc or self-loop (s, s') of $D[S]$ increase the number of disjoint paths between s and s' in D by at least one, Observation 8.4 implies that for every non-arc or self-loop (s, s') of $D[S]$ there are at most $2k$ inner path segments that contribute to (s, s'). Finally, because S has at most k vertices and at most $k(k-1)$ ordered pairs of vertices, we conclude that there are at most $2k^2(k-1)+2k^2$ inner path segments in $D \setminus S$. Having obtained a bound on the number of inner path segments too, we conclude the following.

Observation 8.15. *The number of path segments in $D \setminus S$ is at most* $6k^2(k-1)+4k^2-6$.

Our next aim is to provide a bound on the overall length of path segments and use it to bound the size of A_2. Towards this aim we introduce reduction rules that allow us to bound the in-degree and the out-degree w.r.t. S of any vertex occurring internally in path segments.

Definition 8.16. Let $s \in S$ and let $P = (v_1, \ldots, v_r)$ be an induced directed path in $D \setminus S$, whose internal vertices are in A_2 and that satisfies:

- $(s, v_1) \in E(D)$ and $(s, v_r) \in E(D)$ and v_1 is a balanced vertex in A_2,
- for every i with $1 < i < r$, it holds that $(s, v_i) \notin E(D)$.

If P satisfies the above properties we call P an out-segment for s. We say that P contributes to a potential arc or self-loop (s, s') in $D[S]$ if there is an index i with $1 \leq i < r$ such that $(v_i, s') \in E(D)$ for some $s' \in S$.

We now introduce a reduction rule that allows us to preprocess and reduce certain out-segments.

Reduction Rule 8.7. Let $s \in S$ and let $P = (v_1, \ldots, v_r)$ be an out-segment for s. If P does not contribute to any non-arc or self-loop of $D[S]$, then we remove the arc (s, v_1).

Proof of soundness. Let D' be the directed graph obtained from D after removing the arc (s, v_1). We show that a set S' of size at most k is a DFVS for D if and only if it is a DFVS for D'. The forward direction is trivial because D' is a subgraph of D.

For the backward direction let S' be a DFVS for D' and assume for a contradiction that S' is not a DFVS for D. Then there is a directed cycle C in $D \setminus S'$ that uses the arc (s, v_1). Because v_1 is a balanced vertex the cycle C has to continue on P after using the arc (s, v_1). That is, there is a subpath of C which is $(s, v_1, \ldots, v_q)$ for some $q < r$. We now consider the following two exhaustive cases.

Case 1: *The cycle leaves P at some vertex v_i with* $1 \leq i < r$. Then, C leaves P and enters some vertex $s' \in S$. In other words, C has a subpath $P_C = (s, v_1, \ldots, v_i, s')$. Because P does not contribute to a self-loop of $D[S]$, we have that $s \neq s'$. Moreover, because P does not contribute to a non-arc of $D[S]$ we obtain that $(s, s') \in E(D)$. Hence we can replace the subpath P_C in C with the arc (s, s') and thereby obtain a cycle C' that is also a cycle in $D' \setminus S'$ contradicting our assumption that S' is a DFVS for D'.

Case 2: *The cycle leaves P only at v_r.* In the second case we obtain a cycle C' from C by replacing the arc (s, v_1) plus the segment of C in P with the arc (s, v_r). Because C' is also a cycle in $D' \setminus S'$, this contradicts our assumption that S' is a DFVS for D'.

Having obtained a contradiction in either case, we conclude that S is indeed a DFVS for D. □

After the exhaustive application of the above rule, we obtain the following.

Lemma 8.17. *For each $s \in S$, there are at most k^2 out-segments for s.*

Proof. Since Rule 8.7 does not apply, every out-segment for s contributes to at least one non-arc or self-loop of $D[S]$. Furthermore, every out-segment for s that contributes to some non-arc or self-loop (s, s') of $D[S]$ increases the number of internally vertex-disjoint paths s-s' paths in D by one. Observation 8.4 implies that for every non-arc or self-loop (s, s') of $D[S]$, there are at most k out-segments for s in $D \setminus (S \cup B)$ that contribute to (s, s').

Finally, because every vertex s is contained in at most a single self-loop and in at most $k-1$ non-arcs of $D[S]$, we infer that there are at most $k(k-1)+k$ out-segments for s. □

We are now ready to bound the size of the set A_2.

Lemma 8.18. *The number of vertices in A_2 is at most $12k^4 - 2k^3 - 12k$.*

Proof. We begin by arguing that for every $s \in S$, s has at most $(6k^2(k-1) + 4k^2 - 6 + k(k-1) + k)$ neighbors in A_2. Since the number of out-neighbors of s in A_2 is at most the number of path segments (bounded by Observation 8.15) plus the number of out-segments for s (bounded by Lemma 8.17), we obtain an upper bound of $6k^2(k-1) + 4k^2 - 6 + k^2$ on the size of the out-neighborhood (and by symmetry, the in-neighborhood) of s in A_2.

Consequently, the total number of neighbors of vertices in S to vertices in A_2 and thus (because of Observation 8.10) the total number of vertices in A_2 is at most $2k \cdot (6k^2(k-1) + 4k^2 - 6 + k^2)$. □

We are now ready to prove a bound on the size of the kernel. Recall that thus far we have obtained the following bounds:

- there are at most k^2 vertices in $D \setminus S$ contributing to a self-loop in $D[S]$ (Observation 8.5),
- $|A'_0| \leq k^2(k-1)$ (Observation 8.6),
- $|A'_1| \leq k^2(k-1)$ (see paragraph before Observation 8.8),
- $|A_2| \leq 12k^4 - 2k^3 - 12k$ (Lemma 8.18),
- $|A_{\geq 3}| \leq k^3 - 2$ (Observation 8.8),

It follows that the total number of vertices in the reduced graph is at most $k^2 + |A'_0 \cup A'_1 \cup A_2 \cup A_{\geq 3} \cup S|$ which is at most $k^2 + 2k^2(k-1) + k^3 - 2 + 12k^4 - 2k^3 - 12k + k$, thus proving Theorem 8.1. This completes the description of our kernel for general instances of the problem; we now proceed to the linear kernel on graphs of bounded genus.

8.2 A Linear Kernel for DFVS[FVS] on Bounded Genus graphs

We note that we do not formally define the genus of a graph here. However, to obtain our results, we only need the following result that can be found for example in the book Topological Graph Theory by Gross and Tucker [109]. For a more detailed treatment of topological graph theory the reader is referred for example to the aforementioned book.

Proposition 8.19 ([109]). *The orientable and nonorientable genus, denoted by γ and $\tilde{\gamma}$, of complete bipartite graphs is given by the following formulae:*

$$\gamma(K_{m,n}) \;=\; \left\lceil \frac{(m-2)(n-2)}{4} \right\rceil, m,n \geq 2; \qquad \tilde{\gamma}(K_{m,n}) = \left\lceil \frac{(m-2)(n-2)}{2} \right\rceil, m,n \geq 2.$$

Corollary 8.20. *If G is a graph such that $\gamma(G) \leq g$ and $\tilde{\gamma}(G) \leq h$ for some constants g and h, then G does not contain $K_{3,4g+3}$ nor $K_{3,2h+3}$ as a minor.*

Throughout this section we will use D to denote a directed graph of genus at most some fixed bound g. We let S be a feedback vertex set of $\overline{D}$ and let c be a constant such that $\overline{D}$ is a $K_{3,c}$-minor-free graph, where c depends only on g as per Corollary 8.20. We begin with the following lemma, which follows directly from [92, Lemma 4.3] and from the fact that $\overline{D}$ is $K_{3,c}$-minor-free.

Lemma 8.21. *Let $G = (X, Y, E)$ be a bipartite graph and c a constant such that G is $K_{3,c}$-minor-free. Then,*

- *there are $\mathcal{O}(|X|)$ subsets $X' \subseteq X$ such that $X' = N(u)$ for some $u \in Y$ and*
- *for any subset $X' \subseteq X$ such that $|X'| \geq 3$, the set $Y' = \{y \in Y : N(y) \supseteq X'\}$ has size at most $c - 1$.*

We will need a few additional notions to provide a concise presentation of the results in this section. A digraph H is called a road iff $\overline{H}$ is a path; the first and last vertex on a road are called its *endpoints*, and all other vertices on a road are called *internal* vertices. Moreover, for a directed graph G consider a connected component of $\overline{G}$ with vertex set A such that $G[A]$ is acyclic. Then $G[A]$ (and, equivalently, the set A) is called an acyclic component of G. Since every component of $\overline{D} \setminus S$ is a tree, observe that there is a one-to-one correspondence between connected components of $\overline{D} \setminus S$ and acyclic components of $D \setminus S$.

For each distinct $x, y \in S$, we denote by $\mathcal{C}_{x,y}$ the set of all acyclic components C of $D \setminus S$ with $N(C) = \{x, y\}$. Finally, we use $\mathcal{C}_{x,y}^{\rightarrow}$ to denote the subset of $\mathcal{C}_{x,y}$ of components C with the property that $D[C \cup \{x, y\}]$:

- contains a directed path from x to y, but
- contains neither an x-x directed path nor a y-y directed path intersecting C.

Observe that any road within D that is disjoint from a feedback vertex set of $\overline{D}$ may only contain simple edges. Furthermore, in any instance where Reduction Rule 8.1 is not applicable, the input digraph D itself does not contain an acyclic component. That is, every connected component of $\overline{D}$ contains a directed cycle.

Observation 8.22. *If Reduction Rule 8.1 is not applicable and C is an acyclic component of $D \setminus S$, then there is a directed $N(C)$-$N(C)$ path in $D[C \cup N(C)]$ containing at least one vertex of C.*

Crucially, Observation 8.22 implies that for each component C in $\mathcal{C}_{x,y}$, either $D[C \cup \{x\}]$ (or $D[C \cup \{y\}]$ by symmetry) contains a cycle, or $C \in \mathcal{C}_{x,y}^{\rightarrow} \cup \mathcal{C}_{y,x}^{\rightarrow}$.

Reduction Rule 8.8. If C is an acyclic component of $D \setminus S$ where $N_{\overline{D}}(C) = \{x\}$ for some $x \in S$ and $C \cup \{x\}$ contains a cycle, then we remove $C \cup \{x\}$ from D and reduce k by 1.

The soundness of the above rule follows from the fact that any DFVS which does not contain x must necessarily intersect C, and hence there also exists a DFVS of at most the same size which contains x but does not intersect C.

By expanding the above argument, we observe that if there exists a DFVS T containing at least two vertices from $\mathcal{C}_{x,y} \cup \{x, y\}$, then the set $T' = (T \setminus \mathcal{C}_{x,y}) \cup \{x, y\}$ is also clearly a solution of at most the same size as T. Hence every minimum DFVS contains at most two vertices from $\mathcal{C}_{x,y} \cup \{x, y\}$. Consequently, if we have a minimum DFVS T and $Z_1, Z_2, Z_3 \in \mathcal{C}_{x,y}$, then at least one of Z_1, Z_2 or Z_3 has an empty intersection with T. The soundness of the following three Reduction Rules follows simply from the fact that one of the tree acyclic components is not hit by a minimum-size solution and hence forces the flow in reduction rules.

Reduction Rule 8.9. If $\mathcal{C}_{x,y}$ contains at least 3 acyclic components C_1, C_2, C_3 such that $D[C_1 \cup x]$, $D[C_2 \cup x]$ and $D[C_3 \cup x]$ each contains a cycle, we remove x and decrease k by 1.

Reduction Rule 8.10. If $\mathcal{C}_{x,y}^{\rightarrow} \cap \mathcal{C}_{y,x}^{\rightarrow}$ contains at least 3 components, then we remove all components of $\mathcal{C}_{x,y}^{\rightarrow} \cup \mathcal{C}_{y,x}^{\rightarrow}$ from D and add the arcs (x, y) and (y, x) to D.

Reduction Rule 8.11. If $\mathcal{C}_{x,y}^{\rightarrow}$ contains at least 3 components and $\mathcal{C}_{x,y}^{\rightarrow} \setminus \mathcal{C}_{y,x}^{\rightarrow}$ is not empty, then we remove all components of $\mathcal{C}_{x,y}^{\rightarrow} \setminus \mathcal{C}_{y,x}^{\rightarrow}$ from D and add the arc (x, y) to D.

Lemma 8.23. *After applying Reduction Rule 8.1 and Reduction Rules 8.8 to 8.11, the resulting digraph is also $K_{3,c}$-minor-free.*

Proof. Since Reduction Rule 8.1 and Reduction Rule 8.8 only remove vertices, it clearly does not affect the fact that D is $K_{3,c}$-minor-free. Furthermore, the operations in Reduction Rule 8.10 and Reduction Rule 8.11 result in a graph D' such that $\overline{D'}$ is a minor of $\overline{D}$. Hence, the graph resulting from these reduction rules is also $K_{3,c}$-minor-free. $\square$

We now argue the main structural consequence of applying the reduction rules stated up to this point.

Lemma 8.24. *Suppose that Reduction Rule 8.1 and Reduction Rules 8.8 to 8.11 do not apply. Then $D \setminus S$ has $\mathcal{O}(|S|)$ acyclic components.*

Proof. Let G be the bipartite graph obtained from $\overline{D}$ by leaving only a single copy of all multiple edges, removing all edges between vertices of S, and contracting every connected component of $\overline{D} \setminus S$ into a single vertex. Since G is a minor of $\overline{D}$, it is $K_{3,c}$-minor-free as well; moreover, there is a one-to-one correspondence between connected components of $\overline{D} \setminus S$ and vertices of $G \setminus S$.

We now partition the connected components of $\overline{D} \setminus S$ as follows. For each non-empty $X \subseteq S$, let $\mathcal{T}_X$ be the set of all connected components of $\overline{D} \setminus S$ whose neighborhood is precisely X. By applying Lemma 8.21 to G (with bipartition S and $V(G) \setminus S$), it follows that there are $\mathcal{O}(|S|)$ subsets $X \subseteq S$ such that $\mathcal{T}_X$ is non-empty. Furthermore, for every $X \subseteq S$ of size at least 3, the size of $\mathcal{T}_X$ is at most $c - 1$.

Hence, in order to prove the lemma, it suffices to bound the number of connected components of $\overline{D} \setminus S$ with at most two neighbors in S. In particular, we will show that D does not have 'too many' acyclic components with the same neighborhood of size at most 2. Since there are no sinks or sources in D and no acyclic component of $D \setminus S$ has a single neighbor in S (due to Reduction Rule 8.8), every acyclic component of $D \setminus S$ has at least 2 neighbors in S.

Let us fix distinct $x, y \in S$ and consider the set $\mathcal{C}_{x,y}$. Since Reduction Rule 8.9 is not applicable, there are at most 2 components C in $\mathcal{C}_{x,y}$ such that $D[C \cup \{x\}]$ contains a cycle and analogously for $D[C \cup \{y\}]$. Since Reduction Rule 8.10 is not applicable, $\mathcal{C}^{\rightarrow}_{x,y} \cap \mathcal{C}^{\rightarrow}_{y,x}$ contains at most 2 components. Furthermore, since Reduction Rule 8.11 is not applicable, the sets $\mathcal{C}^{\rightarrow}_{x,y} \setminus \mathcal{C}^{\rightarrow}_{y,x}$ and $\mathcal{C}^{\rightarrow}_{y,x} \setminus \mathcal{C}^{\rightarrow}_{x,y}$ both contain at most 2 components. Finally, from Observation 8.22, it follows that $\mathcal{C}_{x,y}$ does not contain any other components and hence $|\mathcal{C}_{x,y}| \leq 10$. $\square$

Lemma 8.25. *Let C be a connected component of $\overline{D} \setminus S$, and ℓ be the number of neighbors of C in S. If Reduction Rules 8.1 and 8.2 are not applicable, then $\overline{D}[C]$ has $\mathcal{O}(\ell)$ leaves.*

Proof. Recall that when the first two reduction rules do not apply, every vertex in D is incident to at least 4 arcs and thus every leaf of C is incident to at least 3 arcs with endpoints in S. However, since there can be bidirectional arcs between C and S, every leaf of C has at least 2 neighbors in S. From Lemma 8.21 it follows that there are only $\mathcal{O}(\ell)$ vertices in C with at least 3 neighbors in S. Hence it suffices to obtain an $\mathcal{O}(\ell)$ bound on the leaves of C with exactly two neighbors in S.

Recalling Lemma 8.21, we observe that each of the leaves of C has one of $\mathcal{O}(\ell)$ possible neighborhoods in S. Let us fix two distinct vertices x and y in $N(C)$. We will show that there are at most $c-1$ leaves of C with both x and y as neighbors. Suppose for a contradiction that there is a set L of at least c leaves of $\overline{D}[C]$ which are adjacent to x and y. Since $\overline{D}[C]$ is a tree and L is a subset of its leaves, the graph $\overline{D}[C \setminus L]$ is also a tree. If we contract $\overline{D}[C \setminus L]$ into a single vertex, say z, then the subgraph induced on the vertices in $L \cup \{x, y, z\}$ would be isomorphic to $K_{3,c}$, contradicting the fact that $\overline{D}$ is $K_{3,c}$-minor-free. We conclude that C can have at most $c-1$ leaves with neighbors x and y. □

Lemma 8.26. *If none of Reduction Rules 8.1, 8.2, 8.8-8.11 apply, then $\overline{D} \setminus S$ has at most $\mathcal{O}(|S|)$ vertices of degree at least 3.*

Proof. Let $\mathcal{C}$ be the set of all components of $D \setminus S$. Since none of the aforementioned reduction rules apply, we can invoke Lemma 8.24 and Lemma 8.25. That is, we conclude that there are only $\mathcal{O}(|S|)$ components in $\mathcal{C}$ and that the number of leaves in a component $C \in \mathcal{C}$ is $\mathcal{O}(|N(C)|)$.

Since the number of leaves in a tree gives an upper bound on the number of vertices of degree at least 3, this implies that the number of vertices of degree at least 3 in $\overline{D}[C]$ is also bounded by $\mathcal{O}(|N(C)|)$. Thus it suffices to show that $\sum_{C \in \mathcal{C}} |N(C)| = \mathcal{O}(|S|)$. However, $\sum_{C \in \mathcal{C}} |N(C)|$ is the same as the number of edges in the graph G which we obtain from $\overline{D}$ by contracting each component of $\mathcal{C}$ to a single vertex and removing all edges between vertices in S. Since G is clearly a minor of $\overline{D}$, it is also $K_{3,c}$-minor-free and hence $|E(G)|$ is at most $\mathcal{O}(|V(G)|) = \mathcal{O}(|S| + |\mathcal{C}|) = \mathcal{O}(|S|)$. □

The main consequence of the above lemma is that we can now add all the vertices of degree at least 3 to the set S in order to get a set S' which is also a feedback vertex set of $\overline{D}$ of size $\mathcal{O}(|S|) = \mathcal{O}(k)$. At the same time, the graph $\overline{D} \setminus S'$ is significantly more structured: every connected component of this graph is in fact a path and this will play a crucial role in the rest of this section. Since $|S'| \in \mathcal{O}(|S|)$, it suffices to obtain a reduced instance of size linear in $|S'|$, and so for ease of presentation we will hereinafter set $S := S'$.

Reduction Rule 8.12. If none of the Reduction Rules 8.1-8.2, 8.8-8.11 apply, then we add all vertices of total degree three in $D \setminus S$ to S.

Observe that after applying Reduction Rule 8.12, $D \setminus S$ is a set of roads. Furthermore, once we ensure that $D \setminus S$ is a set of roads, none of the reduction rules in this section will ever create a new degree 3 vertex in $D \setminus S$. Hence we can exhaustively apply all the reduction rules in this section once again to ensure that the number of roads in $D \setminus S$ is $\mathcal{O}(|S|)$. In the rest of the section, we present reduction rules to handle the roads in $D \setminus S$.

8.2.1 Dealing with roads

Our first step will be to transform our instance so that all roads in $D \setminus S$ are even more structured with respect to their adjacencies with S.

Definition 8.27. A road P is *nice* if $N(P') \setminus V(P) \subseteq S$ and $|N(P') \setminus V(P)| \leq 2$, where P' are the internal vertices of P.

In other words, nice roads are roads whose internal vertices are *all* adjacent to at most two specific vertices from S (other than the endpoints of the road); observe that this is equivalent to requiring that $|N(P')| \leq 4$, where P' is the set of internal vertices of the nice road P. In order to achieve this transformation, we will iteratively construct an auxiliary vertex set Q to store certain vertices that form separators between nice road segments in $D - S$. In the course of this procedure, we will also construct an injective mapping δ from Q to the connected components of $D \setminus (S \cup Q)$. We initialize by setting $\delta = Q = \emptyset$.

Reduction Rule 8.13. Let A be a connected component which is road in $D \setminus (S \cup Q)$ that is not nice. Moreover, let A' be a maximal nice subroad of A which contains a leaf in $\overline{D} \setminus (S \cup Q)$ and let a' be the unique neighbor of A' in A. Then add a' to Q and add $a' \mapsto A'$ to δ.

For each vertex $q \in Q$, let $R_q = \{q\} \cup \delta(q)$. Observe that R_q is a road which contains at least 3 neighbors in S. Furthermore, for any $q, q' \in Q$ our construction of δ ensures that R_q and $R_{q'}$ are vertex-disjoint.

Lemma 8.28. *After the exhaustive application of Reduction Rule 8.13, we have* $|Q| = \mathcal{O}(|S|)$.

Proof. Let $\mathcal{R}$ be the set $\{R_q | q \in Q\}$ and let G be the graph, which we obtain from $\overline{D}$ by deleting all vertices in $V(D) \setminus (\bigcup_{R \in \mathcal{R}} V(R) \cup S)$, contracting each $R \in \mathcal{R}$ to a single vertex v_R, and deleting all the edges besides the edges between S and v_R for some $R \in \mathcal{R}$. Clearly, G is a minor of $\overline{D}$ and hence $K_{3,c}$-minor-free. Moreover, G is a bipartite graph with partitions S and $T = \{v_R | R \in \mathcal{R}\}$ such that $N(v_R) = N(R) \cup S$ for each $R \in \mathcal{R}$ and hence $|N(v)| \geq 3$ for all vertices v in T. Therefore, it follows from (1) in Lemma 8.21 that there are $\mathcal{O}(|S|)$ different sets $X \subseteq S$ such that $X = N(v)$ for some vertex $v \in T$. Furthermore, from (2) in Lemma 8.21 follows that for each set $X \subseteq S$ there are at most $c - 1$ vertices $v \in T$ such that $X = N(v)$. □

The next rule is only applied *once* after the exhaustive application of Reduction Rule 8.13. Note that it does not increase the parameter by more than a linear factor due to Lemma 8.28.

Reduction Rule 8.14. Set $S := S \cup Q$.

Observe that after the exhaustive application of Reduction Rule 8.13 and the application of Reduction Rule 8.14, each road in $D - S$ is nice. Furthermore, the number of roads in $D - S$ is still linear in S, since removing $|Q|$ vertices from a set of roads only increases the number of roads in the set by at most $|Q|$. Our next task is to deal with nice roads, but we first state a useful observation about general roads.

Observation 8.29. *Let P be a road in $D \setminus S$ and let P' be the internal vertices of P. For any DFVS T of D, the set $(T \setminus P') \cup N(P')$ is also a DFVS of D. In particular, every minimum DFVS contains at most $|N(P')|$ vertices of $P \cup N(P')$.*

Reduction Rule 8.15. Let P be a nice road in $D \setminus S$, P' internal vertices of P, and x a vertex in $N(P') \setminus V(P)$. If $D[P' \cup \{x\}]$ contains at least $|N(P')|$ directed cycles intersecting only in x, then we remove x from D and set $k = k - 1$.

Proof of soundness. In order to prove that this reduction rule is sound, we show that there is always an optimal solution that contains x. Let T be an optimal solution that does not contain the vertex x. Clearly, T contains a vertex for each of at least $|N(P')|$ cycles in $D[P' \cup \{x\}]$ intersecting in x. However, then $T' = (T \setminus P') \cup N(P')$ is also a solution. Moreover, $|T'| \leq |T|$. Hence T' is an optimal solution, which concludes the proof. □

For internal vertices of a road P, we define an equivalence relation $\sim_P$ such as $a \sim_P b$ if and only if $N^+(a) \setminus V(P) = N^+(b) \setminus V(P)$ and $N^-(a) \setminus V(P) = N^-(b) \setminus V(P)$ (i.e., a and b have same out- and in- neighborhoods outside of P). We are now ready to state our final reduction rule, which will later allow us to bound the length of each nice road by a constant.

Reduction Rule 8.16. Let P be a nice road in $D \setminus S$, and let P' be internal vertices of P with $\ell = |N(P')|$. If P' contains a directed subpath $Q = (q_1, \dots, q_{\ell+2})$, such that $q_i \sim_P q_j$ for all $1 \leq i, j \leq \ell + 1$, then we remove q_ℓ from D and add the arc $(q_{\ell-1}, q_{\ell+1})$.

Proof of soundness. We now prove the soundness of this reduction rule. Let D denote the input digraph before application of Reduction Rule 8.16 and D' the digraph after. First, let T' be an optimal solution for D'. We will show that T' is a solution for D as well. Suppose otherwise, and let C be a cycle in $D \setminus T'$. Clearly, C contains the vertex q_ℓ, since otherwise C would lie also in $D' \setminus T'$. There are 4 possibilities for the combination of the predecessor p and successor s of q_ℓ in this cycle.

- $p = q_{\ell-1}$, $s = q_{\ell+1}$

- $p = q_{\ell-1}$, $s \in N^+(q_\ell) \setminus \{q_{\ell+1}\}$
- $p \in N^-(q_\ell) \setminus \{q_{\ell-1}\}$, $s = q_{\ell+1}$
- $p \in N^-(q_\ell) \setminus \{q_{\ell-1}\}$, $s \in N^+(q_\ell) \setminus \{q_{\ell+1}\}$

However, from the definition of Q it follows that in the first 3 cases, there is a (p, s) arc in D' and hence a cycle in $D \setminus T'$ as well, a contradiction. On the other hand, it follows from Observation 8.29 that T' contains at most ℓ vertices of Q, and hence there exists a vertex $q \in Q \setminus (T \cup \{q_\ell\})$. But $q \sim_P q_\ell$, implying that there is a (p, q) arc and a (q, s) arc in D', implying the presence of a directed p-s path (p, q, s) in $D' \setminus T'$. This path can be used in place of the path (p, q_ℓ, s) to construct a cycle from C, which is disjoint from T' in D', a contradiction. Therefore, we conclude that T' is a solution for D as well.

Conversely, let T be an optimal solution for D and let $T' = T$ if T does not contain q_ℓ and $T' = (T \setminus \{q_\ell\}) \cup \{q_{\ell+1}\}$ otherwise. Suppose for contradiction that $D' \setminus T'$ contains a cycle C. Clearly C contains the arc $(q_{\ell-1}, q_{\ell+1})$, since otherwise C is contained also in $D \setminus T$. As a result, it must be the case that T' does not contain $q_{\ell-1}$ and $q_{\ell+1}$, the latter of which can happen only if T does not contain q_ℓ. But then $D \setminus T$ contains a cycle, which we get from C by replacing the arc $(q_{\ell-1}, q_{\ell+1})$ with the directed path $(q_{\ell-1}, q_\ell, q_{\ell+1})$. $\square$

We are now ready to complete the proof of our linear kernelization by bounding the size of an instance after the exhaustive application of our reduction rules.

Lemma 8.30. *If none of Reduction Rules 8.1, 8.2 and 8.8-8.16 apply, then* $|D| = \mathcal{O}(|S|)$.

Proof. Let $\mathcal{P}$ be the set of all acyclic component of $D \setminus S$. Recall that we have already established that since Reduction Rules 8.13 and 8.14 do not apply, every connected component in $\mathcal{P}$ is a nice road in D and there are at most $|\mathcal{P}| = \mathcal{O}(|S|)$ such nice roads. Therefore, it suffices to show that there is a constant d such that every nice road in $\mathcal{P}$ has at most d vertices.

Let us fix a nice road $P \in \mathcal{P}$, with internal vertices P'. Let us denote the vertices of P as $p_1, \ldots, p_t$ such that either (p_i, p_{i+1}) or (p_{i+1}, p_i) is an arc in D for every $1 \leq i < t$. Recall that a road itself is not necessarily a directed path. Since Reduction Rule 8.15 does not apply, there are at most 3 vertices in P with bidirectional arcs to the same vertex of $N(P)$. In particular, that means that if $|N(P') \setminus V(P)| = 1$, then P' has at most 3 vertices. Recall that from the definition of nice roads it follows that $|N(P') \setminus V(P)| \leq 2$. Therefore, in the rest of the proof we assume that $|N(P') \setminus V(P)| = 2$, and we refer to the vertices in this set as x and y.

Since Reduction Rules 8.1 and 8.2 do not apply, it follows that all vertices in P' have at least 2 in-neighbors and 2 out-neighbors. In particular, if $p_i \in P'$ is a sink in $D[V(P)]$, then D contains arcs (p_i, x) and (p_i, y). Similarly, if $p_i \in P'$ is a source in $D[V(P)]$, then D contains arcs (x, p_i) and (y, p_i). Therefore, if for $i < j$ is p_i a source (sink) and p_j a sink (source) in $D[V(P)]$ such that for all $i < k < j$ the vertex p_k is not source nor sink in $D[V(P)]$, then $y, p_i, \ldots, p_j, y$ $(y, p_j, \ldots, p_i, y)$ is a directed cycle. Since sinks and sources have to alternate on the road P and Reduction Rule 8.15 does not apply, there are at most 7 sinks and sources in total in P'. We already showed that P' contains are at most 3 vertices with bidirectional arc to x, 3 vertices with bidirectional arc to y and 7 vertices that are either sink or source in $D[V(P)]$. Let T denote this set of at most 13 vertices of P'. Furthermore, it must be the case that for every $p_i \in P' \setminus T$ either $N^+(p_i) = x$ and $N^-(p_i) = y$, or otherwise $N^+(p_i) = y$ and $N^-(p_i) = x$. We will say that p_i satisfying former condition has Type 1 and the latter condition Type 2.

Note that if for $2 \leq i \leq t-2$ the vertex p_i has Type 1 (Type 2) and p_{i+1} has Type 2 (Type 1), then if D contains the arc (p_i, p_{i+1}), then p_i, p_{i+1}, y (p_i, p_{i+1}, y) is a directed cycle. Otherwise, D contains the arc (p_{i+1}, p_i) and p_{i+1}, p_i, x (p_{i+1}, p_i, y) is a directed cycle. Now let $1 < i < j < t$ be such that $p_i, p_j \in T$ and $p_k \in P' \setminus T$ for all $i < k < j$. It follows that p_k has either Type 1 or Type 2 for all k with $i < k < j$. Moreover, each time p_k and p_{k+1} have distinct types, either $D[\{p_k, p_{k+1}, x\}]$ or $D[\{p_k, p_{k+1}, y\}]$ contains a cycle. Since Reduction Rule 8.15 does not apply, there are can be at most 7 alternations of the type. Finally, since Reduction Rule 8.16 does not apply, there are at most 5 vertices of the same type in a row. It follows that between every pair of consecutive vertices of T on P, there are at most 35 vertices and hence P' contains at most $13 \cdot 35$ vertices. Since P was chosen arbitrarily, the argument holds for every nice road in $D \setminus S$. $\square$

8.3 Summary and Open Questions

Our results provide a stepping stone towards resolving the existence of a polynomial kernel for DFVS, and to the best of our knowledge also represent the first kernelization results for DFVS with respect to any natural parameter. They also open up several new directions for future research. For instance, can we find reasonable parameters that lie "between" DFVS number and FVS number, and would it be possible to generalize our polynomial kernel to these? What about parameters which are incomparable to the FVS number but also upper-bound the DFVS number? Can our linear kernel be lifted to graph classes of bounded expansion or nowhere dense graphs? Another related problem of interest is whether DFVS can be solved in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$, which remains open even on planar graphs.

Notes

The results in this chapter appeared in a conference paper in the proceedings of 43nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017) [21].

CHAPTER 9

Modulators for ILPs

In spite of recent advances [95, 96, 129], we still lack a deep understanding of which structural restrictions make ILP tractable. The goal of this line of research is to identify structural properties (parameters) which allow us to solve ILP efficiently. Here, we initiate the study of distance to triviality for ILP by analyzing modulators which fracture the instance into small, easy-to-handle components. Such *fracture modulators* (also called *fracture backdoors*) can equivalently be viewed as measuring the number of global variables or global constraints in an otherwise "compact" instance; in fact, we identify and analyze three separate cases depending on whether we allow global variables only, global constraints only, or both.

Results

We obtain a near-complete complexity landscape for the considered parameters: in particular, we identify the circumstances under which they can be used to obtain fixed-parameter and XP algorithms for ILP, and otherwise prove that such algorithms would violate well-established complexity assumptions. Our results are summarized in the following Table 9.1 (formal definitions are given in Section 9.2).

	Variable	Constraint	Mixed
param.	FPT (Cor. 9.15)	FPT (Cor. 9.15)	XP (Cor. 9.14)
unary	paraNP-c	XP, W[1]-h	paraNP-c
	(Th 9.23)	(Th 9.22, 9.24)	(Th 9.23)
arbitrary	paraNP-c	paraNP-c (Th 9.25)	paraNP-c

Table 9.1: Complexity landscape for fracture modulators. Columns distinguish whether we consider variable modulators, constraint modulators, or mixed modulators. Rows correspond to restrictions placed on coefficients in the ILP instance.

As is evident from the table, modulator size on its own is not sufficient to break the NP-hardness of ILP; this is far from surprising, and the same situation arose in previous work on treewidth. However, while positive results on treewidth (as well as other considered decomposition parameters such as *torso-width* [96]) required the imposition of domain restrictions on variables, in the case of modulators one can also deal with instances with unrestricted variable domains—by instead restricting the values of coefficients which appear in the ILP instance. Here, we distinguish three separate cases (corresponding to three rows in Table 9.1): coefficients bounded by the parameter value, coefficients which are encoded in unary, and no restrictions. It is worth noting that in the case of treewidth, ILP remains NP-hard even when coefficients are restricted to ± 1 and 0.

Our results in row 1 represent a direct generalization of three extensively studied classes of ILP, specifically n-fold ILP, two-stage stochastic ILP and 4-block N-fold ILP [57, 165]. These classes have been used to obtain faster algorithms for a wide range of problems. For example a range of transportation and logistic problems have been encoded as a two-stage stochastic ILP [173, 120]. Furthermore, some state-of-the art algorithms for some problems in scheduling and computational social choice are obtained by encoding the problem as N-fold ILP [142, 143]. The distinction in our approach lies in the fact that while in the case of all three previously mentioned special cases of ILP the ILP matrix must be completely uniform outside of its global part, here we impose no such restriction. The only part of our complexity landscape which remains incomplete, the case of mixed modulators combined with bounded coefficients, then corresponds to resolving a challenging open problem in the area of N-folds: the fixed-parameter (in)tractability of 4-block N-fold ILP [116]. A fixed-parameter algorithm for 4-block N-fold would also provide significant algorithmic improvements for problems in areas such as social choice [143].

In the intermediate case of coefficient values encoded in unary (row 2), we surprisingly show that ILP remains polynomially tractable when the number of global constraints is bounded by a constant, but becomes NP-hard if we use global variables instead. To be precise, we obtain an XP algorithm parameterized by constraint modulators, rule out the existence of a fixed-parameter algorithm for this case, and also rule out XP algorithms for variable and mixed modulators. These also represent our most technical results: especially the XP algorithm requires the combination of deep linear-algebraic techniques with tools from the parameterized complexity toolbox.

Last but not least, all our algorithmic results first require us to compute a fracture modulator. It turns out that computing fracture modulators in ILP is closely related to solving the Vertex Integrity problem [66] on bipartite graphs; unfortunately, while the problem has been studied on numerous graph classes including cobipartite graphs, its complexity remained open on bipartite graphs. Here we obtain both an exact fixed-parameter algorithm as well as a polynomial time approximation algorithm for finding fracture modulators. As an additional result, we also show that the problem is NP-complete using a novel reduction.

Organization of the Chapter

The chapter is structured as follows. After introducing the Integer Linear Programing in Section 9.1, we proceed to formally define our parameter in Section 9.2 and develop algorithms for computing the desired modulators in Section 9.3. We then present our results separated by the type of restrictions put on the size of the matrix coefficients in the remaining sections.

9.1 Integer Linear Programming

In the following let $\mathbf{A}$ be a $n \times m$ matrix and let C and R be a subset of columns and rows of $\mathbf{A}$, respectively. We denote by $\mathbf{A}_{(R,C)}$ the submatrix of $\mathbf{A}$ restricted to the columns in C and the rows in R. We also denote by $\mathbf{A}_{(*,C)}$ and $\mathbf{A}_{(R,*)}$ the submatrix of $\mathbf{A}$ restricted to the columns in C and the submatrix of $\mathbf{A}$ restricted to the rows in R, respectively. We denote by $c_{\mathbf{A}}$ the maximum absolute value of any entry of $\mathbf{A}$ and by $\det(\mathbf{A})$ the determinant of $\mathbf{A}$. For a vector $\mathbf{b}$ of size n, we will use $\mathbf{b}[i]$ to denote its i-th entry and we denote by $c_{\mathbf{b}}$ the maximum absolute value of any entry of $\mathbf{b}$. We will also use the two following well-known facts [185].

Proposition 9.1. *Let* $\mathbf{A}$ *be an integer* $k \times k$ *matrix. Then* $\det(\mathbf{A})$ *is integer and* $|\det(\mathbf{A})| \leq k!\Pi_{1 \leq i \leq k} c_{\mathbf{A}_{(*,\{i\})}}$.

Proposition 9.2 (Cramer's rule)**.** *Let* $\mathbf{A}$ *be a* $k \times k$ *non-singular (i.e., with non-zero determinant) matrix and* $\mathbf{b}$ *a vector. Then the equation* $\mathbf{A}\mathbf{x} = \mathbf{b}$ *has a unique solution such that* $\mathbf{x}[i] = \frac{\det(\mathbf{A}(i))}{\det(\mathbf{A})}$, *where* $\mathbf{A}(i)$ *is the matrix formed by replacing the* i*-th column of* $\mathbf{A}$ *with the vector* $\mathbf{b}$.

For our purposes, it will be useful to consider ILP instances which are in *equation form*. Formally, let an ILP instance $\mathbf{I}$ be a tuple $(\mathbf{A}, \mathbf{x}, \mathbf{b}, \mathbf{l}, \mathbf{u}, \eta)$, where:

- $\mathbf{A}$ is a $n \times m$ matrix of integers (the *constraint matrix*),
- $\mathbf{x}$ is a vector of *variables* of size m,
- $\mathbf{b}$ is an integer vector of size m (the *right-hand side*),
- $\mathbf{l}, \mathbf{u}$ are vectors of elements of $\mathbb{Z} \cup \{\pm\infty\}$ (the *lower* and *upper bounds*, respectively), and
- η is an integer vector of size m (the *optimization function*).

Let A be the i-th row of $\mathbf{A}$; then we will call $A\mathbf{x} = \mathbf{b}[i]$ a constraint of $\mathbf{I}$. We will use $\text{var}(\mathbf{I})$ to denote the set of *variables* (i.e., the elements of $\mathbf{x}$), and $\mathcal{F}(\mathbf{I})$ (or just $\mathcal{F}$) to denote the set of constraints. For a subset U of $\text{var}(\mathbf{I}) \cup \mathcal{F}(\mathbf{I})$, we denote by $C(U)$ the columns of $\mathbf{A}$ corresponding to variables in U and by $R(U)$ the rows of $\mathbf{A}$ corresponding to constraints in U.

A (partial) assignment α is a mapping from some subset of $\text{var}(\mathbf{I})$, denoted by $\text{var}(\alpha)$, to $\mathbb{Z}$. An assignment α is called *feasible* if

1. satisfies every constraint in $\mathcal{F}$, i.e., if $A\alpha(\mathbf{x}) = \mathbf{b}[i]$ for each i-th row A of $\mathbf{A}$, and
2. satisfies all the upper and lower bounds, i.e., $\mathbf{l}[i] \leq \alpha(\mathbf{x}[i]) \leq \mathbf{u}[i]$.

Furthermore, α is called a *solution* if the value of $\eta\alpha(\mathbf{x})$ is maximized over all feasible assignments; observe that the existence of a feasible assignment does not guarantee the existence of a solution (there may exist an infinite sequence of feasible assignments α with increasing values of $\eta\alpha(\mathbf{x})$; in this case, we speak of *unbounded* instances). Given an instance $\mathbf{I}$, the task in the ILP problem is to compute a solution for $\mathbf{I}$ or correctly determine that no solution exists. We remark that other formulations of ILP exist (e.g., a set of inequalities over variables); it is well-known that these are equivalent and can be transformed into each other in polynomial time [185]. Moreover, such transformations will only change our parameters (defined in Section 9.2) by a constant factor.

Aside from general integer linear programming, we will also be concerned with two subclasses of the problem.

1. ILP-feasibility is formulated equivalently as ILP, with the restriction that η must be the 0-vector. All hardness results for ILP-feasibility immediately carry over to ILP.
2. Unary ILP is the class of all ILP instances which are supplied in a unary bit encoding; in other words, the input size of Unary ILP upper-bounds not only the number of variables and constraints, but also the absolute values of all numbers in the input. Unary ILP remains NP-complete in general, but in our setting there will be cases where its complexity will differ from general ILP.

Combining both restrictions gives rise to Unary ILP-feasibility.

There are several ways of naturally representing ILP instances as graphs. The representation that will be most useful for our purposes will be the so-called incidence graph: the incidence graph $G_{\mathbf{I}}$ of an ILP instance $\mathbf{I}$ is the graph whose vertex set is $\text{var}(\mathbf{I}) \cup \mathcal{F}(\mathbf{I})$ and two vertices s, t are adjacent iff $s \in \text{var}(\mathbf{I})$, $t \in \mathcal{F}$ and s occurs in t with a non-zero coefficient. An instance $\mathbf{I}'$ is a *connected component* of $\mathbf{I}$ if it is the subinstance of $\mathbf{I}$ corresponding to a connected component of $G_{\mathbf{I}}$; formally, $\mathcal{F}(\mathbf{I}') \subseteq \mathcal{F}(\mathbf{I})$ is the set of constraints that occur in a connected component of $G_{\mathbf{I}}$ and $\eta(\mathbf{I}')$ is the restriction of $\eta(\mathbf{I})$ to $\text{var}(\mathcal{F}(\mathbf{I}'))$. For a set $Z \subseteq \mathcal{F}(\mathbf{I}) \cup \text{var}(\mathbf{I})$, we will also use $\mathbf{I} \setminus Z$ to denote the ILP instance obtained by removing all constraints in Z from $\mathcal{F}(\mathbf{I})$ and removing all variables in Z from all constraints in $\mathcal{F}(\mathbf{I}) \setminus Z$ and from η.

For our algorithms, we will use the following result as a subroutine. Note that this is a streamlined version of the original statement of the theorem, as used in the area of parameterized algorithms [79].

Proposition 9.3 ([146, 132, 90])**.** *There is an algorithm that solves an input ILP instance* $\mathbf{I} = (\mathcal{F}, \eta)$ *in time* $p^{\mathcal{O}(p)} \cdot |\mathbf{I}|$*, where* $p = |var(\mathbf{I})|$*.*

9.1.1 ILP with Structured Matrices

Our results build on and extend the classical variable-dimension ILP techniques detailed for instance in the work of [57, 165, 116]. Below, we provide a basic introduction to these techniques and related results. Let $\mathbf{A} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{A}_2 \\ \mathbf{A}_3 & \mathbf{A}_4 \end{pmatrix}$ be a 2×2 block integer matrix. The *N-fold 4-block product of* $\mathbf{A}$ (denoted by $\mathbf{A}^{(N)}$) is the following integer matrix

$$\mathbf{A}^{(N)} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{A}_2 & \mathbf{A}_2 & \cdots & \mathbf{A}_2 \\ \mathbf{A}_3 & \mathbf{A}_4 & 0 & \cdots & 0 \\ \mathbf{A}_3 & 0 & \mathbf{A}_4 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_3 & 0 & 0 & \cdots & \mathbf{A}_4 \end{pmatrix}.$$

Here $\mathbf{A}^{(N)}$ contains N copies of matrices $\mathbf{A}_2, \mathbf{A}_3$, and $\mathbf{A}_4$. Furthermore, $\mathbf{A}_1$ is an $r \times s$ matrix, $\mathbf{A}_2$ is an $r \times t$ matrix, $\mathbf{A}_3$ is an $u \times s$ matrix, and $\mathbf{A}_4$ is an $u \times t$ matrix; for convenience, we let $b_{\mathbf{A}} = \max(r, s, t, u)$. We call an instance $(\mathbf{A}, \mathbf{x}, \mathbf{b}, \mathbf{l}, \mathbf{u}, \eta)$ of ILP an *N-fold 4-block* if $\mathbf{A}$ is an N-fold 4-block product of some 2×2 block integer matrix. Observe that in such instances the vector $\mathbf{x}$ is naturally partitioned into a global part (consisting of s variables) and a local part.

Theorem 9.4 ([116])**.** *Let a and z be constants and let $\mathbf{I}$ be an N-*fold 4-block *ILP instance with $c_{\mathbf{A}} \leq a$, $b_{\mathbf{A}} \leq z$, then $\mathbf{I}$ can be solved in polynomial time.*

In the parameterized complexity setting, the above theorem yields an XP algorithm solving ILP parameterized by $\max(b_{\mathbf{A}}, c_{\mathbf{A}})$ if the matrix is a N-fold 4-block product. We note that the existence of a fixed-parameter algorithm for this problem remains a challenging open problem [116]. However, the problem is known to be fixed-parameter tractable when either $\mathbf{A}_1 = \mathbf{0}$ and $\mathbf{A}_3 = \mathbf{0}$ or $\mathbf{A}_1 = \mathbf{0}$ and $\mathbf{A}_2 = \mathbf{0}$; these variants are called the *N-fold ILP* problem and the *2-stage stochastic ILP* problem, respectively.

Theorem 9.5 ([117], [57])**.** *N-fold ILP and 2-stage stochastic ILP are FPT parameterized by $c_{\mathbf{A}}$ and $b_{\mathbf{A}}$.*

9.2 The Fracture Number

We are now ready to formally introduce the studied parameter and related notions. An ILP instance $\mathbf{I}$ is called *ℓ-compact* if each connected component of $\mathbf{I}$ contains at most ℓ

$$\begin{array}{ll}
\texttt{maximize} \sum_{\texttt{i}=1}^{7} \texttt{i} \cdot \texttt{x}_\texttt{i}, & \text{where} \\
\sum_{\texttt{i}=1}^{7} \texttt{x}_\texttt{i} = 32, & \\
1\texttt{x}_1 + \texttt{y} = 6, & 2\texttt{x}_2 + \texttt{y} = 9, \\
3\texttt{x}_3 + \texttt{y} = 14, & 4\texttt{x}_4 + \texttt{y} = 21, \\
5\texttt{x}_5 + \texttt{y} = 30, & 6\texttt{x}_6 + \texttt{y} = 41.
\end{array}$$

Figure 9.1: The constraints and optimization function of a simple ILP instance with $\mathfrak{p} = 2$, witnessed by a modulator containing y and the first constraint.

variables and constraints; equivalently, each connected component of $G_{\mathbf{I}}$ contains at most ℓ vertices. It is not difficult to observe that any ℓ-compact ILP instance can be solved in time at most $\ell^{\mathcal{O}(\ell)} \cdot |\mathbf{I}|$ due to Proposition 9.3; indeed, we can compute a solution for $\mathbf{I}$ by combining solutions for each connected component of $\mathbf{I}$, and hence it suffices to apply Proposition 9.3 independently on each component.

A set $Z \subseteq \mathcal{F} \cup \text{var}(\mathbf{I})$ is called a *modulator to ℓ-compactness* if $\mathbf{I} \setminus Z$ is ℓ-compact; moreover, if $Z \cap \mathcal{F} = \emptyset$ then Z is called a *variable-modulator to ℓ-compactness*, and if $Z \cap \text{var}(\mathbf{I}) = \emptyset$ then Z is a *constraint-modulator to ℓ-compactness*. We use $b_\ell(\mathbf{I})$ to denote the cardinality of a minimum modulator to ℓ-compactness, and similarly $b_\ell^V(\mathbf{I})$ and $b_\ell^C(\mathbf{I})$ for variable-modulators and constraint-modulators to ℓ-compactness, respectively. It is easy to see that, depending on the instance, $b_\ell^V(\mathbf{I})$ can be arbitrarily larger or smaller than $b_\ell^C(\mathbf{I})$. On the other hand, $b_\ell(\mathbf{I}) \leq \min(b_\ell^V(\mathbf{I}), b_\ell^C(\mathbf{I}))$.

Clearly, the choice of ℓ has a major impact on the size of modulators to ℓ-compactness; in particular, $b_\ell(\mathbf{I})$ could be arbitrarily larger than $b_{\ell+1}(\mathbf{I})$, and the same of course also holds for variable- and constraint-modulators. Since we will be interested in dealing with cases where both ℓ and $b_\ell(\mathbf{I})$ are small, we will introduce the fracture number $\mathfrak{p}$ which provides bounds on both ℓ and b_ℓ; in particular, we let $\mathfrak{p}(\mathbf{I}) = \min_{\ell \in \mathbb{N}}(\max(\ell, b_\ell(\mathbf{I})))$. Furthermore, we say that a modulator witnesses $\mathfrak{p}(\mathbf{I})$ if $|Z| \leq \mathfrak{p}(\mathbf{I})$ and $\mathbf{I} \setminus Z$ is $\mathfrak{p}(\mathbf{I})$-compact. We define $\mathfrak{p}^C(\mathbf{I})$ and $\mathfrak{p}^V(\mathbf{I})$ similarly, with $b_\ell(\mathbf{I})$ replaced by $b_\ell^C(\mathbf{I})$ and $b_\ell^V(\mathbf{I})$, respectively. If the instance $\mathbf{I}$ is clear from the context, we omit the reference to $\mathbf{I}$; see Figure 9.1 for an example.

We remark that the fracture number represents a strict generalization of the parameter $b_{\mathbf{A}}$ used in Theorems 9.4 and 9.5; in particular, $\mathfrak{p} \leq 2b_{\mathbf{A}}$ (and similarly for $\mathfrak{p}^V$ and $\mathfrak{p}^C$ for the latter two theorems). Moreover, the fracture number is well-defined for all ILP instances, not only for N-fold 4-block products. In this respect, N-fold 4-block products with bounded $b_{\mathbf{A}}$ form the subclass of instances with bounded $\mathfrak{p}$ such that each component *must contain precisely the same submatrix.* It is not difficult to see that this is indeed a very strong restriction.

9.3 Computing the Fracture Number

Our evaluation algorithms for ILP require a modulator set as a part of their input. In this section we show how to efficiently compute small modulator sets, i.e., we show how to solve the following problem.

FRACTURE BACKDOOR DETECTION (BD)	
Instance:	An ILP instance $\mathbf{I}$ and a natural number k.
Parameter:	k
Question:	Determine whether $\mathfrak{p}(\mathbf{I}) \leq k$ and if so output a modulator set witnessing this.

We also define the variants V-BD and C-BD that are concerned with finding a variable or a constraint modulator, respectively, in the natural way. Observe that at its core the above problem and its variants are really a problem on the incidence graph of the ILP instance. Namely, the problems can be equivalently stated as the following graph problem.

FRACTURE VERTEX DELETION (FVD)	
Instance:	An undirected bipartite graph G with bipartition $\{U, W\}$, a set $D \in \{U, V(G)\}$, and an integer k.
Parameter:	k
Question:	Is there a set $B \subseteq D$ of at most k vertices such that every connected component of $G \setminus B$ has size at most k?

It is worth noting that this graph problem is closely related to the so-called VERTEX INTEGRITY problem, which has been studied on a variety of graph classes, including co-bipartite graphs [66]. Unfortunately, to the best of our knowledge nothing is known about its complexity on bipartite graphs.

To see that each variant of BD is equivalent to a specific subcase of the FVD problem (in particular depending on the choice of D in the instance), consider the following polynomial time reductions in both directions. Given an instance $(\mathbf{I}, k)$ of BD, then the instance $(G_{\mathbf{I}}, V(G_{\mathbf{I}}), k)$ of FVD is easily seen to be equivalent. Similarly, if $(\mathbf{I}, k)$ is an instance of V-BD or C-BD, then $(G_{\mathbf{I}}, \text{var}(\mathbf{I}), k)$ and $(G_{\mathbf{I}}, \mathcal{F}(\mathbf{I}), k)$ are equivalent instances of FVD. Moreover, if $I = (G, V(G), k)$ is an instance of FVD, then $(\mathbf{I}, k)$, where $\mathbf{I}$ is any ILP instance such that $G_{\mathbf{I}}$ is isomorphic with G is an equivalent instance of BD. Similarly, if $I = (G, U, k)$ is an instance of FVD, then $(\mathbf{I}, k)$, where $\mathbf{I}$ is any ILP instance such that $G_{\mathbf{I}}$ is isomorphic with G and $\text{var}(\mathbf{I}) = U$, is an equivalent instance of V-BD. Note that such an instance $\mathbf{I}$ can for instance be obtained as follows:

- for every vertex $v \in U$, $\mathbf{I}$ has one variable v with arbitrary domain,

- for every vertex $v \in W$, $\mathbf{I}$ has one constraint with arbitrary non-zero coefficients on the variables in $N_G(v)$,

To justify a parameterized complexity analysis of our detection problems, we first show NP-completeness of our problems. It is worth noting that the NP-completeness of Fracture Vertex Deletion was far from obvious at first glance due to the restriction to bipartite graphs; indeed, for instance the related problem of deleting at most k vertices such that the remaining graph only contains isolated vertices (Vertex Cover) is well-known to be polynomial on bipartite graphs. We note that BD and its variants are closely related to the so-called Vertex Integrity problem on bipartite graphs [66]. We use this connection to obtain our complexity result for BD and as a side result show that Vertex Integrity is NP-complete even on bipartite graphs, which has not been known so far.

Theorem 9.6. BD*,* V-BD*, and* C-BD *are* NP*-complete.*

The proof follows from a reduction from 3-SAT where every literal occurs in exactly two clauses; this variant is well-known to be NP-complete [98]). At its core, the reduction utilizes variable gadgets as well as clause gadgets, and the main difficulty lies in designing these in order to ensure that the graph remains bipartite.

Proof of Theorem 9.6. Because of the equivalence between BD, V-BD, C-BD and the FVD problem, it is sufficient to show that FVD is NP-complete for both choices of D. Because any solution to FVD can be verified in polynomial time, it holds that FVD is in NP. Towards showing NP-hardness of FVD we give a polynomial time reduction from a known variant of the 3-Satisfiability problem. Given a 3-CNF formula Φ with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$ such that every literal occurs in exactly two clauses (this variant of 3-Satisfiability is known to be NP-complete [98]), we construct the instance $\langle G, D, k\rangle$ of FVD as follows. We set $k = n + 2m$ and the graph G will be the disjoint union of certain variable and clause gadgets introduced below plus connections between these variable and clauses gadgets. Namely, for every variable x_i, the graph G contains the variable gadget $G(x_i)$ with the following vertices and edges:

- two vertices x_i and $\overline{x_i}$,
- $k-5$ vertices $c_i^1, \ldots, c_i^{k-5}$,
- for every j with $1 \leq j \leq k-5$ the two edges $\{x_i, c_i^j\}$ and $\{\overline{x_i}, c_i^j\}$.

Moreover for every clause C_j of Φ with literals l_j^1, l_j^2, l_j^3, the graph G contains a clause gadget $G(C_J)$ with the following vertices and edges:

- three vertices l_j^1, l_j^2, and l_j^3,

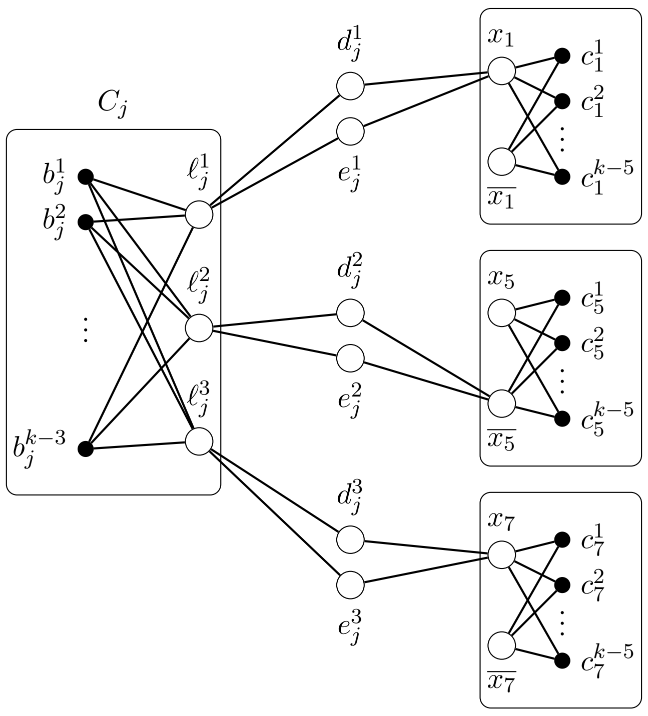


Figure 9.2: The interaction between clause and vertex gadgets for the clause $C_j = x_1 \vee \overline{x_5} \vee x_7$.

- $k-3$ vertices $b_j^1, \dots, b_j^{k-3}$,
- for every i with $1 \leq i \leq k-3$ the three edges $\{b_j^i, l_j^1\}$, $\{b_j^i, l_j^2\}$, and $\{b_j^i, l_j^3\}$.

Note that $G(C_j)$ is simple a complete bipartite graph with bipartition $\{\{l_j^1, l_j^2, l_j^3\}, \{\, b_j^i \mid 1 \leq i \leq k-3\,\}\}$. Now G consists of the disjoint union of $G(x_1), \dots, G(x_n)$, $G(C_1), \dots, G(C_m)$ plus the following vertices and edges, which ensure the required connections between the variable and clause gadgets (see Figure 9.2 for an example):

- For every clause C_j (for some j with $1 \leq j \leq m$) with literals l_j^1, l_j^2, and l_j^3 and every $a \in \{1, 2, 3\}$ we add the vertices d_j^a and e_j^a and the edges $\{l_j^a, d_j^a\}$ and $\{l_j^a, e_j^a\}$ to G. Moreover, if $l_j^a = x_i$ for some i with $1 \leq i \leq n$, we additionally add the edges $\{x_i, d_j^a\}$ and $\{x_i, e_j^a\}$ to G and if on the other hand $l_j^a = \overline{x_i}$ for some i as above, then we add the edges $\{\overline{x_i}, d_j^a\}$ and $\{\overline{x_i}, e_j^a\}$ to G.

This completes the construction of G, which is clearly bipartite as for instance witnessed by the bipartition $\{U, V(G) \setminus U\}$, where $U = \{\, x_i, \overline{x_i}, l_j^1, l_j^2, l_j^3 \mid 1 \leq i \leq n \wedge 1 \leq j \leq m \,\}$. We will show below that there is always a solution that is entirely contained in U, which implies that the hardness result holds for $D \in \{U, V(G)\}$, and hence all versions of the fracture modulator set problem, i.e., BD, V-BD, and C-BD, are NP-complete. Note that the reduction can be computed in polynomial time and it remains to show the equivalence between the two instances.

Towards showing the forward direction, assume that $\alpha : \{x_1, \dots, x_n\} \to \{0, 1\}$ is a satisfying assignment for Φ. Because α satisfies Φ it follows that for every clause C_j with literals l_j^1, l_j^2, and l_j^3 there is at least one index $a(C_j) \in \{1, 2, 3\}$ such that the literal $l_j^{a(C_j)}$ is satisfied by α. We claim that the set B defined by:

- for every i with $1 \leq i \leq n$, B contains x_i if $\alpha(x_i) = 1$ and $\overline{x_i}$, otherwise,
- for every j with $1 \leq j \leq m$, B contains the vertices in $\{\, l_j^b \mid b \in \{1, 2, 3\} \setminus \{a(C_j)\}\,\}$.

is a solution for (G, U, k). Because B contains exactly one vertex for every variable of Φ and exactly two vertices for every clause of Φ, it holds that $|B| = k = n + 2m$, as required. Moreover, $B \subseteq U$. It hence only remains to show that every component of $G \setminus B$ has size at most k. Towards showing this first consider a component C of $G \setminus B$ that contains at least one vertex from a variable gadget $G(x_i)$ for some i with $1 \leq i \leq n$. Then $G(x_i) \cap B \in \{\{x_i, \overline{x_i}\}\}$ and hence $G(x_i) \setminus B$ is connected, which implies that $G(x_i) \setminus B \subseteq C$. W.l.o.g., assume that $G(x_i) \cap B = \{x_i\}$. Then $\alpha(x_i) = 1$ and it follows that all literal vertices of clause gadgets that correspond to the literal $\overline{x_i}$ are contained in B. Since moreover $\overline{x_i}$ is contained in exactly two clauses, we obtain that C consists of exactly $k - 4$ vertices in $G(x_i) \setminus B$ plus the four vertices d_{j1}^{a1}, e_{j1}^{a1}, d_{j2}^{a2} and e_{j2}^{a2} defined by $l_{j1}^{a1} = \overline{x_i}$ and $l_{j2}^{a2} = \overline{x_i}$. Hence in total C contains exactly k vertices as required. Now consider a component C that contains at least one vertex from a clause gadget $G(C_j)$ for some j with $1 \leq j \leq m$. Then $|G(C_j) \cap B| = 2$ and moreover B contains all but exactly one literal vertex say l_j^a for some $a \in \{1, 2, 3\}$ from $G(C_j)$. W.l.o.g., let x_i be the literal of C_j corresponding to l_j^a. Then $\alpha(x_i) = 1$ and hence $x_i \in B$. It follows that C consists of the exactly $k - 2$ vertices in $G(C_j) \setminus B$ plus the two vertices d_j^a and e_j^a. Hence in total C contains exactly k vertices, as required. Because every component of $G \setminus B$ that neither contains a vertex from a vertex gadget nor from a clause gadget has size exactly one, this shows that B is indeed a solution for (G, U, k) and hence also for $(G, V(G), k)$.

Towards showing the reverse direction, let B be a solution for $(G, V(G), k)$. We first show that, w.l.o.g., we can assume that $B \subseteq U$. So assume that $B \nsubseteq U$. We distinguish three cases: B contains a vertex d_j^a or e_j^a for some j and a with $1 \leq j \leq m$ and $1 \leq a \leq 3$. Let u and v be the two vertices adjacent to d_j^a and e_j^a. If B contains both d_j^a and e_j^a, then it is straightforward to verify that $B \setminus \{d_j^a, e_j^a\} \cup \{u, v\}$ is also a solution. So assume that B contains only d_j^a (the case that B contains only e_j^a is analogous). If $\{u, v\} \subseteq B$, then $B \setminus \{d_j^a\}$ is still a solution. Hence assume that, w.l.o.g., $u \notin B$. But then $(B \setminus \{d_j^a\}) \cup \{u\}$ is a solution. Hence in all cases we could transform B into a solution that does not contain a vertex d_j^a or e_j^a. Next consider the case that B contains some vertex c_i^j for some i and j with $1 \leq i \leq n$ and $1 \leq j \leq k - 5$. In this case one can use an argumentation very similar to the previous case to transform B into a solution not containing such a vertex.

Hence there only remains the case that B contains some vertex b_j^i for some i and j with $1 \leq i \leq k-3$ and $1 \leq j \leq m$. In this case it is straightforward to verify that removing all vertices from $B \cap \{b_j^1, \ldots, b_j^{k-3}\}$ and replacing those with an equal (or less) amount of vertices in $\{l_j^1, l_j^2, l_j^3\}$ will again give a solution. Hence we can assume that $B \subseteq U$.

We show next that B contains at least one of x_i and $\overline{x_i}$ from every variable gadget $G(x_i)$. Suppose not and consider the component C of $G \setminus B$ containing x_i. Because $B \subseteq U$, we obtain that C contains all $k-3$ vertices in $G(x_i)$ and additionally at least the 8 vertices adjacent to x_i and $\overline{x_i}$. Hence $|C| \geq k-3+8 > k$ a contradiction to our assumption that B is a solution.

We show next that B contains at least two of $\{l_j^1, l_j^2, l_j^3\}$ from every clause gadget $G(C_j)$. Suppose not and consider a component C of $G \setminus B$ containing at least one vertex from $G(C_j)$. Because $B \subseteq U$, we obtain that C contains all of the at least $k-3+2=k-1$ vertices in $G(C_j) \setminus B$ and additionally the at least four vertices adjacent to the (at least two) literal vertices in $\{l_j^1, l_j^2, l_j^3\} \setminus B$. Hence $|C| \geq k-1+4 > k$ a contradiction to our assumption that B is a solution.

Hence B contains at least one vertex for every variable of Φ and at least two vertices for every clause of Φ. Moreover, because B is a solution it holds that $|B| \leq k = n+2m$. Hence $|B| = n+2m$ and B contains exactly one vertex from every variable gadget and exactly two vertices from every clause gadget. We claim that the assignment α with $\alpha(x_i) = 1$ if and only if $x_i \in B$ is a satisfying assignment for Φ. Suppose not and let C_j be a clause of Φ that is not satisfied by Φ and let l_j^a be the (unique) literal vertex of $G(C_j)$ that is not in B. Consider the component C of $G \setminus B$ that contains l_j^a and assume, w.l.o.g., that $l_j^a = x_i$ for some i with $1 \leq i \leq n$. Because α does not satisfy C_j, we obtain that $x_i \notin B$. Because furthermore $B \subseteq U$ we obtain that C contains all of the $k-3+1=k-2$ vertices in $G(C_j) \setminus B$ and additionally at least the two vertices adjacent to l_j^a as well as the vertex x_i. Hence in total C contains at least $k-2+3 > k$ vertices, a contradiction to our assumption that B is a solution. $\square$

Even though BD is NP-complete, here we provide two efficient algorithms for solving it: we show that the problem is fixed-parameter tractable parameterized by k and can be approximated in polynomial time within a factor of k. Both of these algorithms are based on the observation that any modulator has to contain at least one vertex from every connected subgraph of the instance of size $k+1$.

Theorem 9.7. BD, V-BD, *and* C-BD *can be solved in time* $\mathcal{O}((k+1)^k|E(G)|)$ *and are hence FPT.*

Proof. Because of the equivalence of the problems BD, V-BD, and C-BD with the FVD problem, it is sufficient to show the result for FVD.

We will show the lemma by providing a depth-bounded search tree algorithm for any instance $I = \langle G, D, k \rangle$ of FVD, which is based on the following observations.

O1 If G is not connected then a solution for I can be obtained as the disjoint union of solutions for every component of G.

O2 If G is connected and C is any set of $k+1$ vertices of G such that $G[C]$ is connected, then any solution for I has to contain at least one vertex from C.

These observations lead directly to the following recursive algorithm that given an instance $I = \langle G, D, k\rangle$ of FVD either determines that the instance is a No-instance or outputs a solution $B \subseteq D$ of minimal size for I. The algorithm also remembers the maximum size of any component in a global constant c, which is set to k for the whole duration of the algorithm. The algorithm first checks whether G is connected. If G is not connected the algorithm calls itself recursively on the instance $(C, D \cap C, k)$ for each component C of G. If one of the recursive calls returns No or if the size of the union of the solutions returned for each component exceeds k, the algorithm returns that I is a No-instance. Otherwise the algorithm returns the union of the solutions returned for each component of G.

If G is connected and $|V(G)| \leq c$, the algorithm returns the empty set as a solution. Otherwise, i.e., if G is connected but $|V(G)| > c$ the algorithm first computes a set C of $c+1$ vertices of G such that $G[C]$ is connected. This can for instance be achieved by a depth-first search that starts at any vertex of G and stops as soon as $c+1$ vertices have been visited. If $C \cap D = \emptyset$ then the algorithm returns No. Otherwise the algorithm branches on the vertices in $C \cap D$, i.e., for every $v \in C \cap D$ the algorithm recursively computes a solution for the instance $(G \setminus \{v\}, k-1)$. It then returns the solution of minimum size returned by any of those recursive calls, or No-if none of those calls return a solution. This completes the description of the algorithm. The correctness of the algorithm follows immediately from the above observations. Moreover the running time of the algorithm is easily seen to be dominated by the maximum time required for the case that at each step of the algorithm G is connected. In this case the running time can be obtained as the product of the number of branching steps times the time spent on each of those. Because at each recursive call the parameter k is decreased by at least one and the number of branching choices is at most $c+1$, we obtain that there are at most $(c+1)^k = (k+1)^k$ branching steps. Furthermore, the time at each branching step is dominated by the time required to check whether G is connected, which is linear in the number of edges of G. Putting everything together, we obtain $\mathcal{O}((k+1)^k|E(G)|)$ as the total time required by the algorithm, which completes the proof of the lemma. □

We note that the depth-first search algorithm in the above proof can be easily transformed into a polynomial time approximation algorithm for BD and its variants that exhibits an approximation ratio of $k+1$. In particular, instead of branching on the vertices of a connected subgraph C of G with $k+1$ vertices, this algorithm would simply add all the vertices of C into the current solution. This way we obtain:

Theorem 9.8. *For each of* BD*,* V-BD*, and* C-BD*, there exists a polynomial time algorithm that either finds a (variable-,constraint-) modulator to k-compactness of size at most k^2+k or correctly outputs that the corresponding fracture number is at least $k+1$.*

9.4 The Case of Bounded Coefficients

The goal of this section is to obtain the algorithmic results presented on the first row of Table 9.1. Recall that in this case we will be parameterizing also by $c_{\mathbf{A}}$, which is the maximum absolute value of a coefficient occurring in $\mathbf{A}$. Before we proceed to the results themselves, we first need to introduce a natural notion of "equivalence" among the components of an ILP instance.

Let Z be a modulator to ℓ-compactness for an ILP instance $\mathbf{I}$. We define the equivalence relation $\sim$ on the components of $\mathbf{I} \setminus Z$ as follows: two components C_1 and C_2 are equivalent iff there exists a bijection γ between $\text{var}(C_1)$ and $\text{var}(C_2)$ such that the ILP instance obtained from $\mathbf{I}$ after renaming the variables in $\text{var}(C_1)$ and $\text{var}(C_2)$ according to γ and γ^{-1}, respectively, is equal to $\mathbf{I}$. In other words, if for all $x \in \text{var}(C_1)$ we swap the columns corresponding to x and $\gamma(x)$ in $\mathbf{A}$, we obtain the same instance $\mathbf{I}$. We say that components C_1 and C_2 have the same *type* if $C_1 \sim C_2$.

Lemma 9.9. *Let $\mathbf{I}$ be an ILP instance and $k = \mathfrak{p}(\mathbf{I})$. For any modulator witnessing $\mathfrak{p}(\mathbf{I})$, $\sim$ has at most $\left(2c_{\mathbf{A}}(\mathbf{I}) + 1\right)^{2k^2}$ equivalence classes. Moreover, one can test whether two components have the same type in time $\mathcal{O}(k!k^2)$.*

Proof. Let Z be the modulator witnessing $\mathfrak{p}(\mathbf{I})$ and fix a component C of $\mathbf{I} \setminus Z$. First observe that there are only 3 submatrices of the constraint matrix of $\mathbf{I}$ that can contain nonzero coefficients and containing an element of C; we refer to Fig. 9.3, where we denote these matrices $\mathbf{Q}_C, \mathbf{Q}_C^V$, and $\mathbf{Q}_C^C$. We will now bound the number of these possibly nonzero coefficients. In order to do this we denote by $g_v = |\text{var}(Z)|$, $g_c = |Z| - g_v$, $c_v = |\text{var}(C)|$, $c_c = |C| - c_v$, and $c = \max\{c_c, c_v\}$. Observe that $c \leq k$. Now the number of possibly nonzero coefficients is bounded by $g_v c_c + g_c c_v + c_c c_v \leq (g_v + g_c)c + c^2 \leq 2k^2$. We finish the proof of the first part by observing that the number of possible coefficients is bounded by $2c_{\mathbf{A}}(\mathbf{I}) + 1$.

Observe that two components C_1 and C_2 have the same type if their number of constraints and variables is the same and there exist a permutation of variables of C_1 and a permutation of constraints of C_1 such that the three submatrices of $\mathbf{I}$ containing nonzero elements are exactly the same. Again as $|C| \leq k$ one can check all pairs of permutations in time $k!$ and for each pair we are checking $\mathcal{O}(k^2)$ entries. ☐

We now proceed to the main tool used for our algorithms.

Theorem 9.10. *Let $\overline{\mathbf{I}}$ be an ILP instance with matrix $\overline{\mathbf{A}}$, Z be a modulator set witnessing $\mathfrak{p}(\overline{\mathbf{I}})$, and let n be the number of components of $\overline{\mathbf{I}} \setminus Z$. There is an algorithm which runs in time $\mathcal{O}(n^2(\mathfrak{p}(\mathbf{I}) + 1)! + |\mathbf{I}|)$ and computes an $(r + u) \times (s + t)$ matrix $\mathbf{A} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{A}_2 \\ \mathbf{A}_3 & \mathbf{A}_4 \end{pmatrix}$, a positive integer $N \leq n$, and a 4-block N-fold instance $\mathbf{I} = (\mathbf{A}^{(N)}, \mathbf{x}, \mathbf{b}, \mathbf{l}, \mathbf{u}, \eta)$ such that:*

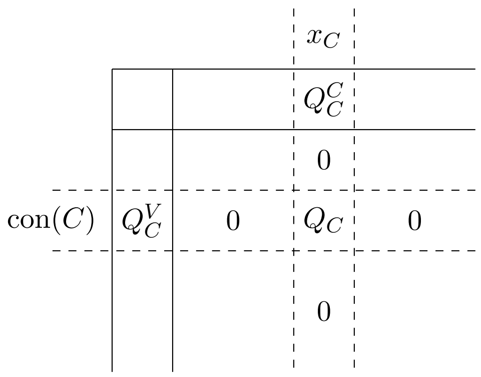


Figure 9.3: A situation for a component C.

(P1) any solution for $\mathbf{I}$ *can be transformed (in polynomial time) into a solution for* $\overline{\mathbf{I}}$ *(and vice versa), and*

(P2) $\max\{r, s\} \leq \mathfrak{p}(\overline{\mathbf{I}})$ *and* $\max\{t, u\} \leq f(c_{\overline{\mathbf{A}}}, \mathfrak{p}(\overline{\mathbf{I}}))$ *for some computable function f.*

Proof. Let $\mathcal{C}$ be the set of connected components of $\overline{\mathbf{I}} \setminus Z$. We define a triple of matrices $(\mathbf{Q}^V_C, \mathbf{Q}^C_C, \mathbf{Q}_C)$ for a component C. Please refer to Figure 9.3.

- The matrix $\mathbf{Q}^V_C$ is the part of constraints in C dealing with variables in Z, that is, $\overline{\mathbf{A}}_{\mathcal{F}(C),\text{var}(Z)}$,
- the matrix $\mathbf{Q}^C_C$ is the part of the constraints in Z dealing with var(C), that is, $\overline{\mathbf{A}}_{\mathcal{F}(Z),\text{var}(C)}$, and
- the matrix $\mathbf{Q}_C$ is the part of constraints in C dealing with var(C), that is, $\overline{\mathbf{A}}_{cF(C),\text{var}(C)}$.

Observe that this totally decomposes all constraints and variables contained in C as all coefficient for other variables are 0 and variables of C cannot appear in other components. For a triple of matrices $T = (\mathbf{Q}^V, \mathbf{Q}^C, \mathbf{Q})$ a component C has type T if $\mathbf{Q}^V = \mathbf{Q}^V_C, \mathbf{Q}_C = \mathbf{Q}^C_C$, and $\mathbf{Q} = \mathbf{Q}_C$ holds. The set of all possible types is the set

$$\mathcal{T} = \{\, T = (\mathbf{Q}^V, \mathbf{Q}^C, \mathbf{Q}) \mid \exists C \in \mathcal{C} \text{ with type } T \,\}.$$

The multiplicity mult(T) of type $T \in \mathcal{T}$ is the number of components in $\mathcal{C}$ having type T. We set $N = \max_{T\in\mathcal{T}} \text{mult}(T)$.

The idea of the proof is to build the matrix $\mathbf{A}_1$ from Z and matrices $\mathbf{A}_2, \mathbf{A}_3, \mathbf{A}_4$ as representatives of the types in such a way that the resulting N-fold 4 block ILP is equivalent to the given ILP instance $\overline{\mathbf{I}}$.

The matrix $\mathbf{A}_1$ is simply the submatrix of Z that is the part of global constraints of $\overline{\mathbf{A}}$ containing var(Z) only.

Claim 9.11. *There is an ILP instance $\hat{\mathbf{I}}$ that is equivalent to ILP instance $\bar{\mathbf{I}}$ with* $\mathrm{mult}_{\hat{\mathbf{I}}}(T) = N$ *for all $T \in \mathcal{T}_{\hat{\mathbf{I}}}$. Moreover, $c_{\hat{\mathbf{I}}} = c_{\bar{\mathbf{I}}}$ and the sizes of the matrices $\mathbf{Q}$ can only double.*

In this case we put all possible matrices on a diagonal of the relevant matrix $\mathbf{A}_4$, next to each other in the matrix $\mathbf{A}_2$, and under each other in the matrix $\mathbf{A}_3$. That is we set $\mathbf{A}_2$ to horizontal concatenation of all $(\mathbf{Q}_T^C)_{T\in\mathcal{T}}$, $\mathbf{A}_3$ to vertical concatenation of $(\mathbf{Q}_T^V)_{T\in\mathcal{T}}$, and finally $\mathbf{A}_4$ has matrices $(\mathbf{Q}_T)_{T\in\mathcal{T}}$ on its diagonal. The bound on size of the matrix $\mathbf{A}$ follows from Lemma 9.9 and Claim 9.11. □

Proof of Claim 9.11. The idea here is to take a type with less representatives and add a new one as a copy of a previous one. But this has to be done carefully in order to maintain equivalence of intermediate ILPs. For the local part we start by observing that if we add a copy of some previous component, then the set of solutions for these two components is the same. However, as these components also interact with the global constraints we would like to have to restrict the set of solutions of the newly added component to all 0 solution only. Note that this cannot be done using lower and upper bounds only as the former set of solutions does not have to contain such a solution. That is, the (optimal) setting of global variables together with setting all component local variables to 0 can violate the right-hand side. In order to achieve the claim, we extend the matrices we have obtained from the component C in the following way. Let C be of type $T = (\mathbf{Q}_C^V, \mathbf{Q}_C^C, \mathbf{Q}_C)$ then the *extension of type T* if

$$\hat{T} = (\mathbf{Q}_C^V, [\mathbf{Q}_C^C \mid \mathbf{0}], [\mathbf{Q}_C \mid \mathbf{Q}_C]).$$

We denote the former C-variables as x_C and the new C-variables as $\hat{x}_C$. We say that the extension is of

- *first kind* if $\ell_C \le x_C \le u_C$ and $0 \le \hat{x}_C \le 0$, and
- *second kind* if $0 \le x_C \le 0$ and $\ell_C \le \hat{x}_C \le u_C$.

Note that with this we have only doubled the number of local variables of component C.

Claim 9.12. *Let $\mathbf{I}$ be an ILP instance and let T be a type of $\mathbf{I}$. Denote $\mathbf{I}_{T\to\hat{T}}$ the ILP instance $\mathbf{I}$ where components of type T are replaced with components of $\hat{T}$ of the first kind. Then, there is a bijection between solutions of ILP instances $\mathbf{I}$ and $\mathbf{I}_{T\to\hat{T}}$.*

Proof of the Claim. Note that it holds that $\hat{x}_C = 0$ for every component C of type $\hat{T}$. Now a solution for $\mathbf{I}_{T\to\hat{T}}$ has a natural projection to a solution of $\mathbf{I}$ (forget all $\hat{x}_C$ variables). Furthermore, a solution for $\mathbf{I}$ can be extended to a solution of $\mathbf{I}_{T\to\hat{T}}$ by setting $\hat{x}_C = 0$ for each component C of type T. This yields a bijection between the solution sets. ◊

We say that a component C is *extended* if it has been created by the extension of the first kind. We transform all components with multiplicity less than N to extended components and denote $\mathbf{I}_E$ the resulting ILP instance. Note that by Claim 9.12 the ILP instances $\mathbf{I}$ and $\mathbf{I}_E$ are in equivalent.

Claim 9.13. *Let $\mathbf{I}$ be an ILP instance, let C be a component of $\mathbf{I}$, and let C' be an extension of C of the second kind. Denote $\mathbf{I}'$ the ILP instance $\mathbf{I}$ with C' added (i.e., it has one more component) then instances $\mathbf{I}$ and $\mathbf{I}'$ are equivalent.*

Proof of the Claim. First we argue that $\mathbf{I}$ does have a solution if and only if $\mathbf{I}'$ does. To see this take a solution $\mathbf{x}$ of $\mathbf{I}$ and let x_C be the part of $\mathbf{x}$ corresponding to C-variables. We build a solution to $\mathbf{I}'$ follows. We copy the solution of every variable but the variables of C'. We set variables $x_{C'} = 0$ and $\hat{x}_{C'} = x_C$.

Note that by this we have actually build a natural correspondence between the set of solutions to $\mathbf{I}$ and the set of solutions to $\mathbf{I}'$. Observe that this correspondence is not one-to-one as in general there can be more possibilities how to extend the solution to variables $\hat{x}_{C'}$. We say that all these solutions project to the same solution $\mathbf{x}$ to instance $\mathbf{I}$. However, as all the C'-variables do not occur in the objective function the value of the objective function of all solutions that project to $\mathbf{x}$ is the same. $\diamondsuit$

By combining the two claims it is possible to transform ILP instance $\mathbf{I}$ to $\hat{\mathbf{I}}$ with the following properties.

- all components of $\hat{\mathbf{I}}$ are either extended or for their type T it holds that $\mathrm{mult}_{\mathbf{I}}(T) = N$,
- for each type $\hat{T}$ of $\hat{\mathbf{I}}$ it holds that $\mathrm{mult}_{\hat{\mathbf{I}}}(\hat{T}) = N$,
- $b_\ell(\hat{\mathbf{I}}) = b_\ell(\mathbf{I})$,
- the number of variables in $\hat{\mathbf{I}}$ is at most twice the number of variables in $\mathbf{I}$. $\square$

Let us now discuss algorithmic consequences of Theorem 9.10 for all types of modulators. Together with Theorems 9.4 and 9.7, we obtain the following corollary.

Corollary 9.14. *Let a and z be constants and let $\mathbf{I}$ be an ILP instance with $c_{\mathbf{A}}(\mathbf{I}) \le a$ and $\mathfrak{p}(\mathbf{I}) \le z$, then $\mathbf{I}$ can be solved in polynomial time.*

For variable and constraint modulators, using Theorem 9.5 instead of Theorem 9.4 yields the following results.

Corollary 9.15. *ILP is FPT when parameterized by $\max\{c_{\mathbf{A}}, \mathfrak{p}^V\}$ and also when parameterized by $\max\{c_{\mathbf{A}}, \mathfrak{p}^C\}$.*

9.5 Unary ILP

Here we will prove that Unary ILP is polynomial time solvable when $\mathfrak{p}^C$ is bounded by a constant; this contrasts the case of general ILP, which remains NP-hard in this case (see Theorem 9.25 later). In particular, we will give an XP algorithm for Unary ILP parameterized by $\mathfrak{p}^C$. We will also present lower bounds showing that such an algorithm cannot exist for Unary ILP parameterized by $\mathfrak{p}^V$ or $\mathfrak{p}$, and rule out the existence of a fixed-parameter algorithm for $\mathfrak{p}^C$.

9.5.1 The Algorithm

The crucial, and also most technically demanding, part of this result is showing that it suffices to restrict our search space to assignments over polynomially bounded variable domains.

Before showing this we need some preparation.

Proposition 9.16. *Let* $\mathbf{A}$ *be an integer* $k \times k$ *non-singular matrix and* $\mathbf{b}$ *an integer vector. Then* $|\mathbf{x}[i]| \leq k! c_{\mathbf{b}}(c_{\mathbf{A}})^{k-1}$ *for the unique* $\mathbf{x}$ *such that* $\mathbf{A}\mathbf{x} = \mathbf{b}$.

Proof. Because of Proposition 9.2 it holds that

$$\mathbf{x}[i] = \frac{\det(\mathbf{A}(i))}{\det(\mathbf{A})}.$$

Moreover, since $\mathbf{A}$ is a non-singular integer matrix, we have that $|\det(\mathbf{A})| \geq 1$ and thus $|\mathbf{x}[i]| \leq |\det(\mathbf{A}(i))|$ and $|\mathbf{x}[i]| \leq |\det(\mathbf{A}(i))| \leq k! c_{\mathbf{b}}(c_{\mathbf{A}})^{k-1}$ follows from applying Proposition 9.1, as required. $\square$

Lemma 9.17. *Let* $\mathbf{Q}$ *be a* $k \times n$ *matrix of rank* k*,* $\mathbf{y}$ *be a vector of* n *variables,* $\mathbf{d}$ *be a vector of size* k*,* I *be a set of* k *linearly independent columns of* $\mathbf{Q}$*,* V *be their corresponding variables in* $var(\mathbf{y})$*, and let* β *be an assignment of the variables in* $var(\mathbf{y})$ *such that* $\mathbf{Q}\beta(\mathbf{y}) = \mathbf{d}$*. Then for every* $v \in V$*, it holds that*

$$|\beta(v)| \leq k!\Big(c_{\mathbf{d}} + c_{\mathbf{Q}} \sum_{u \in var(\mathbf{y}) \setminus V} \beta(u)\Big)(c_{\mathbf{Q}})^{k-1}.$$

Proof. Let $\mathbf{y}'$ be $\mathbf{y}$ restricted to the variables in $\text{var}(\mathbf{y}) \setminus V$ and let J be the set of all columns of $\mathbf{Q}$ that are not in I. We will now apply the assignment β for the variables in $\mathbf{y}'$ to $\mathbf{Q}$. This will give us a set of equations that need to be satisfied for the variables in V allowing us to obtain a bound on β for these variables. Namely, the right-hand side denoted by $\mathbf{d}'$ of our equations is obtained from $\mathbf{d}$ by subtracting the application of β to $\mathbf{Q}_{(*,J)}$, i.e., $\mathbf{d}' = \mathbf{d} - \mathbf{Q}_{(*,J)}\beta(\mathbf{y}')$, which after restricting $\mathbf{Q}$ to the columns I and using the restriction $\mathbf{y}''$ of $\mathbf{y}$ to the variables in V gives us the following equations that are satisfied by β:

$$\mathbf{Q}_{(*,I)}\beta(\mathbf{y}'') = \mathbf{d}' \tag{9.1}$$

Note that because I is a set of k linearly independent columns the matrix $\mathbf{Q}_{(*,I)}$ is non-singular. Moreover, observe that $\mathbf{d}'[i] \leq c_{\mathbf{d}} + c_{\mathbf{Q}} \sum_{u \in \text{var}(\mathbf{y}) \setminus V} \beta(u)$ for every i with $1 \leq i \leq k$. Because β satisfies 9.1 we obtain from Proposition 9.16 that

$$|\beta(v)| \leq k!\Big(c_{\mathbf{d}} + c_{\mathbf{Q}} \sum_{u \in \text{var}(\mathbf{y}) \setminus V} \beta(u)\Big)(c_{\mathbf{Q}})^{k-1},$$

for every variable $v \in V$. ☐

The following lemma provides an important ingredient for Lemma 9.20 below. Its proof crucially makes use of the specific structure of our ILP instance.

Lemma 9.18. *Let* $\mathbf{I}$ *be an instance of* Unary ILP *with matrix* $\mathbf{A}$*. Then for any set* D *of linearly dependent columns of* $\mathbf{A}$*, it holds that* $\mathbf{A}_{(*,D)}$ *contains a subset of at most* $\mathfrak{p}^C(\mathbf{I})(\mathfrak{p}^C(\mathbf{I})+1)$ *linearly dependent columns.*

Proof. Let $Z \subseteq \mathcal{F}(\mathbf{I})$ be a constraint modulator for $\mathbf{I}$ of size at most $\mathfrak{p}^C(\mathbf{I})$ and let $\mathbf{s}$ be a non-zero vector satisfying $\mathbf{A}_{(*,D)}\mathbf{s} = \mathbf{0}$. Let $C_1, \dots, C_p$ be all components of $\mathbf{I} \setminus Z$ that contain at least one variable corresponding to a column in D and let D_i be the set of all columns in D that correspond to variables in C_i. Moreover, let $\mathbf{s}_{C_i}$ be the restriction of $\mathbf{s}$ to the entries corresponding to variables in C_i. Note that if $p \leq \mathfrak{p}^C(\mathbf{I}) + 1$, then D already contains at most $\mathfrak{p}^C(\mathbf{I})(\mathfrak{p}^C(\mathbf{I})+1)$ linearly dependent columns and the lemma follows. So we can assume in the following that $p > \mathfrak{p}^C(\mathbf{I}) + 1$. Denote by $\mathbf{w}_{C_i}$ the vector $\mathbf{A}_{(*,D_i)}\mathbf{s}_{C_i}$. If $\mathbf{w}_{C_i} = \mathbf{0}$, then the variables in C_i that $\mathbf{s}$ does not assign to 0 correspond to at most $\mathfrak{p}^C(\mathbf{I})$ linearly dependent columns and the lemma follows. Otherwise, it is easy to observe that if $\mathbf{w}_{C_i}[j] \neq 0$ then j corresponds to a constraint in Z. Hence for every C_i all non-zero entries of the vector $\mathbf{w}_{C_i}$ correspond to constraints in Z. Consequently any subset of $\mathfrak{p}^C(\mathbf{I})+1$ vectors from $\mathbf{w}_{C_1}, \dots, \mathbf{w}_{C_p}$ in particular the vectors $\mathbf{w}_{C_1}, \dots, \mathbf{w}_{C_{\mathfrak{p}^C(\mathbf{I})+1}}$ are linearly dependent (since all their non-zero entries correspond to constraints in Z and $|Z| \leq \mathfrak{p}^C(\mathbf{I})$), which implies that the set $\bigcup_{1 \leq i \leq \mathfrak{p}^C(\mathbf{I})+1} D_i$ is the required subset of at most $\mathfrak{p}^C(\mathbf{I})(\mathfrak{p}^C(\mathbf{I})+1)$ linearly dependent columns of $\mathbf{A}_{(*,D)}$. ☐

Lemma 9.19. *Let* $\mathbf{I} = (\mathbf{A}, \mathbf{x}, \mathbf{b}, \mathbf{l}, \mathbf{u}, \eta)$ *be an ILP instance,* α *a solution for* $\mathbf{I}$*, and* δ *a non-zero integer vector such that* $\alpha + \delta$ *and* $\alpha - \delta$ *are feasible assignments for* $\mathbf{I}$*. Then* $\eta\delta = 0$ *and moreover* $\alpha + \delta$ *and* $\alpha - \delta$ *are also solutions for* $\mathbf{I}$*.*

Proof. Assume for a contradiction that $\eta\delta \neq 0$, then either $\eta(\alpha+\delta) > \eta(\alpha)$ or $\eta(\alpha-\delta) > \eta(\alpha)$, contradicting that α is a solution. ☐

We are now ready to show that we only need to consider solutions with polynomially bounded variable domain.

Lemma 9.20. *Let* $\mathbf{I}$ *be a feasible instance of* Unary ILP-Feasibility *of size* n. *Then, there exists a solution* α *with* $|\alpha(v)| \leq m_L$ *for every* $v \in \mathit{var}(\mathbf{I})$, *where*

$$m_L = 8(2(\mathfrak{p}^C(\mathbf{I}) + 2)^2)!(n)^{2(\mathfrak{p}^C(\mathbf{I})+2)^2}.$$

Proof. Let $\mathbf{I} = (\mathbf{A}, \mathbf{x}, \mathbf{b}, \mathbf{l}, \mathbf{u}, \eta)$ be the provided instance of Unary ILP and let $Z \subseteq \mathcal{F}(\mathbf{I})$ be a constraint modulator witnessing $\mathfrak{p}^C(\mathbf{I})$.

Let

$$m_{\mathrm{S}} = ((\mathfrak{p}^C(\mathbf{I}) + 1)^2)!(n)^{(\mathfrak{p}^C(\mathbf{I})+1)^2},$$

$$m_{\mathrm{M}} = 4((\mathfrak{p}^C(\mathbf{I}) + 2)^2)!(n)^{(\mathfrak{p}^C(\mathbf{I})+2)^2}.$$

For a solution α of $\mathbf{I}$, let $V(\alpha)$ be the set of all variables v of $\mathbf{I}$ such that $|\alpha(v)| \geq 2m_{\mathrm{S}}$. Let us now consider a solution α which minimizes the size of $V(\alpha)$. Observe that because $m_{\mathrm{L}} \geq 2m_{\mathrm{S}}$ it holds that if $|V(\alpha)| = 0$ then the lemma holds, and so we may assume that $V(\alpha)$ is non-empty.

In the following we consider the submatrix $\mathbf{B} = \mathbf{A}_{(*,V(\alpha))}$. Let us first consider the case where the columns of $\mathbf{B}$ are linearly dependent. We show that in this case, we can find a solution α' such that $|V(\alpha')| < |V(\alpha)|$, which contradicts the choice of α.

Because of Lemma 9.18 there is a non-empty set O of linearly dependent columns of $\mathbf{B}$ of size at most $\mathfrak{p}^C(\mathbf{I})(\mathfrak{p}^C(\mathbf{I})+1)$. Consider a subset $Y = \{\mathbf{v}^1, \dots, \mathbf{v}^{|Y|}\}$ of linearly dependent columns of O such that the columns of each proper subset of Y are linearly independent and let $X = Y \setminus \{\mathbf{v}^{|Y|}\}$. Because Y is a minimal set of linearly dependent columns, it holds that there is a vector $\mathbf{a}$ without any zero entries such that $\mathbf{B}_{(*,Y)}\mathbf{a} = \mathbf{0}$, which implies the existence of a vector $\mathbf{a}_X$, again without zero entries, such that $\mathbf{B}_{(*,X)}\mathbf{a}_X = \mathbf{v}^{|Y|}$. We will show that there is a vector $\mathbf{a}$ that is integer and satisfies $|\mathbf{a}[i]| \leq m_{\mathrm{S}}$ for every $1 \leq i \leq |Y|$. We start by solving $\mathbf{B}_{(*,X)}\mathbf{a}_X = \mathbf{v}^{|Y|}$ using Cramer's rule. Because the columns in X are linearly independent, it follows that $\mathbf{B}_{(*,X)}$ has a set R of linearly independent rows with $|R| = |X|$. Then because the matrix $\mathbf{B}_{(R,X)}$ is non-singular, we have that there is a unique $\mathbf{a}_X$ such that $\mathbf{B}_{(R,X)}\mathbf{a}_X = \mathbf{v}^{|Y|}_R$, where $\mathbf{v}^{|Y|}_R$ denotes the restriction of the vector $\mathbf{v}^{|Y|}$ to the entries associated with the columns in R. Moreover, because there is a non-zero vector $\mathbf{a}_X$ with $\mathbf{B}_{(*,X)}\mathbf{a}_X = \mathbf{v}^{|Y|}$, it follows that the unique vector $\mathbf{a}_X$ satisfying $\mathbf{B}_{(R,X)}\mathbf{a}_X = \mathbf{v}^{|Y|}_R$ also satisfies $\mathbf{B}_{(*,X)}\mathbf{a}_X = \mathbf{v}^{|Y|}$. Using Cramer's Rule, we obtain $\mathbf{a}_X[i] = \frac{\det(\mathbf{B}_{(R,X)}(i))}{\det(\mathbf{B}_{(R,X)})}$ for every i with $1 \leq i \leq |X|$ as the unique vector satisfying $\mathbf{B}_{(R,X)}\mathbf{a}_X = \mathbf{v}^{|Y|}_R$.

Hence the vector $\mathbf{d}$ with $\mathbf{d}[i] = \mathbf{a}_X[i]\det(\mathbf{B}_{(R,X)}) = \det(\mathbf{B}_{(R,X)}(i))$ for every i with $1 \leq i \leq |X|$ and $\mathbf{d}[|Y|] = -\det(\mathbf{B}_{(R,X)})$ is a non-zero integer vector that satisfies $\mathbf{B}_{(*,Y)}\mathbf{d} = \mathbf{0}$. From Proposition 9.1, we obtain that

$$\begin{aligned}
&|\mathbf{d}[i]| \\
&\quad \leq (\mathfrak{p}^C(\mathbf{I})(\mathfrak{p}^C(\mathbf{I})+1))!(c_{\mathbf{A}})^{\mathfrak{p}^C(\mathbf{I})(\mathfrak{p}^C(\mathbf{I})+1)} \\
&\quad \leq ((\mathfrak{p}^C(\mathbf{I})+1)^2)!(n)^{(\mathfrak{p}^C(\mathbf{I})+1)^2} \\
&\quad = m_{\mathrm{S}},
\end{aligned}$$

as required.

For notational convenience we will in the following assume that $\mathbf{A}$ starts with the columns $\mathbf{v}^1, \dots, \mathbf{v}^{|Y|}$ from Y. Let $\mathbf{w}$ be the vector defined by:

- $\mathbf{w}[i] = \mathbf{d}[i]$, if $i \leq |Y|$, and
- $\mathbf{w}[i] = 0$ otherwise

Note that $\mathbf{A}\mathbf{w} = \mathbf{0}$. For an integer Δ, let $\alpha_\Delta : \mathrm{var}(\mathbf{I}) \to \mathbb{Z}$ denote the assignment $\alpha_\Delta = \alpha + \Delta\mathbf{w}$. Note that α_Δ is an integral assignment, moreover because

$$\mathbf{A}\alpha_\Delta(\mathbf{x}) = \mathbf{A}\alpha(\mathbf{x}) + \Delta\mathbf{A}\mathbf{w} = \mathbf{A}\alpha(\mathbf{x})$$

it follows that α_Δ is a feasible integral assignment for $\mathbf{A}\mathbf{x} = \mathbf{b}$ for every $\Delta \in \mathbb{Z}$. Let Δ be the integer with smallest absolute value such that there is at least one variable $v \in V(\alpha)$ with $|\alpha_\Delta(v)| \leq 2m_{\mathrm{S}}$. We claim that for every $|\delta| \leq |\Delta|$, α_δ is a solution for $\mathbf{I}$. We first show that $\mathbf{l}[i] \leq \alpha_\delta(\mathbf{x}[i]) \leq \mathbf{u}[i]$ for every i with $1 \leq i \leq |\mathrm{var}(\mathbf{I})|$. If $\mathbf{x}[i]$ corresponds to a column that is not in Y, then $\alpha_\delta(\mathbf{x}[i]) = \alpha(\mathbf{x}[i])$, which implies $\mathbf{l}[i] \leq \alpha_\delta(\mathbf{x}[i]) \leq \mathbf{u}[i]$. Otherwise, assume, w.l.o.g., that $\alpha(\mathbf{x}[i]) \geq 0$ (the case that $\alpha(\mathbf{x}[i]) < 0$ is symmetric). Because $\alpha(\mathbf{x}[i]) \geq 2m_{\mathrm{S}}$ and $|\mathbf{d}[j]| \leq m_{\mathrm{S}}$ for every j with $1 \leq j \leq |Y|$ together with the choice of Δ, we obtain that $m_{\mathrm{S}} \leq \alpha_\delta(\mathbf{x}[i])$. Because $m_{\mathrm{S}} > n$ and since α is a feasible solution it follows that $\mathbf{u}[i] = \infty$ and $\mathbf{l}[i] \leq m_{\mathrm{S}}$, which shows that $\mathbf{l}[i] \leq \alpha_\delta(\mathbf{x}[i]) \leq \mathbf{u}[i]$. Hence in particular α_Δ and also $\alpha_1 = \alpha + \mathbf{w}$ and $\alpha_{-1} = \alpha - \mathbf{w}$ are feasible assignments, which together with Lemma 9.19 (after setting δ to $\mathbf{w}$) implies that $\eta\mathbf{w} = 0$ and hence $\eta\alpha = \eta\alpha_\Delta$. Consequently α_Δ is a solution for $\mathbf{I}$ with $|V(\alpha_\Delta)| < |V(\alpha)|$, contradicting our choice of α.

We conclude that the columns of $\mathbf{B}$ must be linearly independent, which implies that there is a set R of $|V(\alpha)|$ linearly independent rows in $\mathbf{B}$. Consider the set S of all components of $\mathbf{I} \setminus Z$ that have a non-empty intersection with either $V(\alpha)$ or the constraints corresponding to the rows in R. Let $C_1, \dots, C_p$ be the restrictions of the components in S to the variables in $V(\alpha)$ and the constraints in R.

Observe that for every component C_i, it holds that the rows in R that correspond to constraints in C_i are zero everywhere but at the entries corresponding to variables in C_i. Because the rows in R are independent it follows that every component must have at least as many variables as constraints. Moreover, because $\mathbf{B}_{(R,*)}$ is a square matrix and the only rows in R that do not correspond to constraints in components, correspond to the constraints in Z, we obtain that there are at most $|Z| \leq \mathfrak{p}^C(\mathbf{I})$ components that have strictly more variables than constraints, all other components have the same number of rows and columns. Let C_i be a component with the same number of rows as columns and let C_i' be the unique component of $\mathbf{I} \setminus Z$ containing C_i. Let $Q = \mathbf{A}_{(C(C_i'),R(C_i))}$ and $\mathbf{y}$ be the subvector of $\mathbf{x}$ restricted to the variables of C_i', $\mathbf{d}$ be the subvector of $\mathbf{b}$ restricted to entries that correspond to the constraints of C_i, $V = \text{var}(C_i)$, I the set of columns of Q corresponding to the variables in V, and β the assignment α restricted to the variables in $\mathbf{y}$. Because the rows in Q are independent its rank is $|\mathcal{F}(C_i)|$, because α satisfies $\mathbf{A}\alpha(\mathbf{x}) = \mathbf{b}$ and all but the columns corresponding to the variables in $\text{var}(C_i')$ of $\mathbf{A}_{*,\mathcal{F}(C_i)}$ are zero everywhere, it holds that $\mathbf{Q}\beta(\mathbf{y}) = \mathbf{d}$. Hence we can apply Lemma 9.17 for $\mathbf{Q}$, $\mathbf{y}$, $\mathbf{d}$, V, I, and β and obtain:

$$\begin{aligned}
|\alpha(v)| \leq & \quad \mathfrak{p}^C(\mathbf{I})!\Big(c_{\mathbf{b}} + c_{\mathbf{A}} \sum_{u \in \text{var}(C_i') \setminus \text{var}(C_i)} \alpha(u)\Big)(c_{\mathbf{A}})^{\mathfrak{p}^C(\mathbf{I})-1} && (9.2)\\
\leq & \quad \mathfrak{p}^C(\mathbf{I})!(c_{\mathbf{b}} + c_{\mathbf{A}}\mathfrak{p}^C(\mathbf{I})2m_{\text{S}})(c_{\mathbf{A}})^{\mathfrak{p}^C(\mathbf{I})-1} && (9.3)\\
\leq & \quad \mathfrak{p}^C(\mathbf{I})!4m_{\text{S}}\mathfrak{p}^C(\mathbf{I})(n)^{\mathfrak{p}^C(\mathbf{I})} && (9.4)\\
\leq & \quad 4((\mathfrak{p}^C(\mathbf{I})+2)^2)!(n)^{(\mathfrak{p}^C(\mathbf{I})+2)^2} && (9.5)\\
= & \quad m_{\text{M}} && (9.6)
\end{aligned}$$

for every variable $v \in V$. Inequality 9.3 follows because $|\alpha(v)| \leq 2m_{\text{S}}$ for every v in $\text{var}(C_i') \setminus \text{var}(C_i)$, which is because $(\text{var}(C_i') \setminus \text{var}(C_i)) \subseteq (\text{var}(\mathbf{I}) \setminus V(\alpha))$. This shows that the assignment α is bounded by m_{M} for all variables contained in components C_i that have the same number of variables and constraints. Consider the remaining components $D_1, \dots, D_s$ among $C_1, \dots, C_p$, i.e., the components among $C_1, \dots, C_p$ that have more variables than constraints. Recall that $s \leq |Z| \leq \mathfrak{p}^C(\mathbf{I})$. Let $V = \bigcup_{1 \leq i \leq s} \text{var}(D_i)$ and let J be the corresponding columns of V in $\mathbf{A}$. Note that $|J| \leq (\mathfrak{p}^C(\mathbf{I}))^2$. Because $V \subseteq V(\alpha)$ it holds that J is a set of linearly independent columns. Hence there is a set R' of $|J|$ linearly independent rows in $\mathbf{A}_{(*,J)}$.

Let $\mathbf{Q} = \mathbf{A}_{(R',*)}$, $\mathbf{y} = \mathbf{x}$, $\mathbf{d}$ be the subvector of $\mathbf{b}$ restricted to entries that correspond to the rows in R', I be the columns in J restricted to the rows in R', and $\beta = \alpha$. Because the rows in $\mathbf{Q}$ are independent its rank is $|I|$, because $\mathbf{Q}$ is a submatrix of $\mathbf{A}$ only restricted in rows, we have $\mathbf{Q}\beta(\mathbf{y}) = \mathbf{d}$. Hence we can apply Lemma 9.17 for $\mathbf{Q}$, $\mathbf{y}$, $\mathbf{d}$, V, I, and β and obtain:

$$\begin{aligned}
|\alpha(v)| \leq & \quad (\mathfrak{p}^C(\mathbf{I})^2)!\Big(c_{\mathbf{b}} + c_{\mathbf{A}} \sum_{u \in \text{var}(\mathbf{I}) \setminus V} \alpha(u)\Big)(c_{\mathbf{A}})^{((\mathfrak{p}^C(\mathbf{I}))^2-1)} && (9.7)\\
\leq & \quad (\mathfrak{p}^C(\mathbf{I})^2)!(c_{\mathbf{b}} + c_{\mathbf{A}}|\text{var}(\mathbf{I})|m_{\text{M}})(c_{\mathbf{A}})^{((\mathfrak{p}^C(\mathbf{I}))^2-1)} && (9.8)\\
\leq & \quad 8(2(\mathfrak{p}^C(\mathbf{I})+2)^2)!(n)^{2(\mathfrak{p}^C(\mathbf{I})+2)^2} && (9.9)\\
= & \quad m_{\text{L}} && (9.10)
\end{aligned}$$

for every variable $v \in V$. Inequality 9.8 follows because $|\alpha(v)| \leq m_{\text{M}}$ for every v in $\text{var}(\mathbf{I}) \setminus V$, as shown previously. ☐

To complete the proof of the desired statement, we use a recent result of [96, Proposition 2 and Theorem 11] on solving ILP using treewidth (which is always at most $\mathfrak{p}$) and obtain:

Proposition 9.21 (Proposition 2 and Theorem 11 in [96])**.** *Let* $\mathbf{I} = (\mathbf{A}, \mathbf{x}, \mathbf{b}, \mathbf{l}, \mathbf{u}, \eta)$ *be an ILP with incidence treewidth* ω *and such that* $\mathbf{l}[i] \neq -\infty$ *and* $\mathbf{u}[i] \neq \infty$ *for every entry* i*. Then* $\mathbf{I}$ *can be solved in time* $\mathcal{O}((c_{\mathbf{A}} \cdot \Delta \cdot |var(\mathbf{I})|)^{\omega})(|var(\mathbf{I})| + |\mathcal{F}(\mathbf{I})|)$*, where* $\Delta = \max_i\{|\mathbf{l}[i]|, |\mathbf{u}[i]|\}$*.*

Theorem 9.22. Unary ILP *is polynomial time solvable for any fixed value of* $\mathfrak{p}^C(\mathbf{I})$*, where* $\mathbf{I}$ *is the input instance.*

Proof. Let $\mathbf{I}$ be an input instance of Unary ILP encoded in n bits and let $\mathbf{I}'$ be the instance obtained from $\mathbf{I}$ by replacing $-\infty$ and ∞ entries in $\mathbf{l}$ and $\mathbf{u}$ with $-m_{\mathrm{L}}$ and m_{L}, respectively (for the definition of m_{L} see the statement of Lemma 9.20). It follows from Lemma 9.20 that $\mathbf{I}$ and $\mathbf{I}'$ are equivalent ILP instances. Now let ω be the incidence treewidth of $\mathbf{I}'$ (which is equal to the incidence treewidth of $\mathbf{I}$). Observe that $\omega \leq \mathfrak{p}^C(\mathbf{I})$ and hence it follows from Proposition 9.21 that $\mathbf{I}'$ (and thus also $\mathbf{I}$) can be solved in time $\mathcal{O}((c_{\mathbf{A}} \cdot m_{\mathrm{L}} \cdot |var(\mathbf{I})|)^{\mathfrak{p}^C(\mathbf{I})})(|\mathrm{var}(\mathbf{I})| + |\mathcal{F}(\mathbf{I})|)$. □

9.5.2 Lower Bounds

We complement our algorithm with matching lower bounds: strong NP-hardness for variable and mixed modulators, W[1]-hardness in the case of constraint modulators, and weak NP-hardness for constraint and mixed modulators.

Theorem 9.23. Unary ILP-feasibility *is* paraNP*-hard parameterized by* $\mathfrak{p}^V(\mathbf{I})$*.*

Proof. We prove the theorem by a polynomial time reduction from the well-known NP-hard 3-Colorability problem [98]: given a graph, decide whether the vertices of G can be colored with three colors such that no two adjacent vertices of G share the same color.

The main idea behind the reduction is to represent a 3-partition of the vertex set of G (which in turn represents a 3-coloring of G) by the domain values of three "global" variables. The value of each of these global variables will represent a subset of vertices of G that will be colored using the same color. To represent a subset of the vertices of G in terms of domain values of the global variables, we will associate every vertex of G with a unique prime number and represent a subset by the value obtained from the multiplication of all prime numbers of vertices contained in the subset. To ensure that the subsets represented by the global variables correspond to a valid 3-partition of G we will introduce constraints which ensure that:

C1 For every prime number representing some vertex of G exactly one of the global variables is divisible by that prime number. This ensures that every vertex of G is assigned to exactly one color class.

C2 For every edge $\{u, v\}$ of G it holds that no global variable is divisible by the prime numbers representing u and v at the same time. This ensures that no two adjacent vertices of G are assigned to the same color class.

Thus let G be the given instance of 3-Coloring and assume that the vertices of G are uniquely identified as elements of $\{1, \dots, |V(G)|\}$. In the following we denote by $p(i)$ the i-th prime number for any positive integer i, where $p(1) = 2$. We construct an instance **I** of ILP-feasibility in polynomial time with $\mathfrak{p}^V(\mathbf{I}) \leq 25$, and coefficients bounded by a polynomial in $V(G)$ such that G has a 3-coloring if and only if **I** has a feasible assignment. This instance **I** has the following variables:

- The *global variables* c_1, c_2, and c_3 with an arbitrary positive domain, whose values will represent a valid 3-Partitioning of $V(G)$.
- For every i and j with $1 \leq i \leq |V(G)|$ and $1 \leq j \leq 3$, the variables $m_{i,j}$, $\mathrm{sl}^1_{i,j}$, and $\mathrm{sl}^2_{i,j}$ (with an arbitrary non-negative domain), $r_{i,j}$ (with domain between 0 and $p(i) - 1$), and $u_{i,j}$ (with binary domain). These variables are used to secure condition C1.
- For every $e \in E(G)$, $i \in e$, and j with $1 \leq j \leq 3$, the variables $m_{e,i,j}$, $\mathrm{sl}^3_{e,i,j}$, $\mathrm{sl}^4_{e,i,j}$, and $\mathrm{sl}^5_{e,j}$ (with an arbitrary non-negative domain), $r_{e,i,j}$ (with domain between 0 and $p(i) - 1$), and $u_{e,i,j}$ (with binary domain). These variables are used to secure condition C2.

Note that the variables $\mathrm{sl}^1_{i,j}$, $\mathrm{sl}^2_{i,j}$, $\mathrm{sl}^3_{e,i,j}$, $\mathrm{sl}^4_{e,i,j}$, and $\mathrm{sl}^5_{e,i}$ are so-called "Slack" variables, whose sole purpose is to obtain an ILP instance that is in equation normal form. The instance **I** has the following constraints (in the following let α be any feasible assignment of **I**):

- domain restrictions for all variables as given above, i.e.:
  - for every i and j with $1 \leq i \leq |V(G)|$ and $1 \leq j \leq 3$, the constraints $c_j \geq 0$, $m_{i,j} \geq 0$, $\mathrm{sl}^1_{i,j} \geq 0$, $\mathrm{sl}^2_{i,j} \geq 0$, $0 \leq r_{i,j} \leq p(i) - 1$, and $0 \leq u_{i,j} \leq 1$.
  - for every $e \in E(G)$, $i \in e$, and j with $1 \leq j \leq 3$, the constraints $m_{e,i,j} \geq 0$, $\mathrm{sl}^3_{e,i,j} \geq 0$, $\mathrm{sl}^4_{e,i,j} \geq 0$, $\mathrm{sl}^5_{e,j} \geq 0$, $0 \leq r_{e,i,j} \leq p(i) - 1$, and $0 \leq u_{e,i,j} \leq 1$.
- The following constraints, introduced for each $1 \leq i \leq |V(G)|$ and $1 \leq j \leq 3$, together guarantee that condition C1 holds:
  - Constraints that ensure that $\alpha(r_{i,j})$ is equal to the remainder of $\alpha(c_j)$ divided by $p(i)$, i.e., the constraint $c_j = p(i)m_{i,j} + r_{i,j}$.
  - Constraints that ensure that $\alpha(u_{i,j}) = 0$ if and only if $\alpha(r_{i,j}) = 0$, i.e., the constraints $u_{i,j} + \mathrm{sl}^1_{i,j} = r_{i,j}$ and $r_{i,j} + \mathrm{sl}^2_{i,j} = (p(i) - 1)u_{i,j}$. Note that together the above constraints now ensure that $\alpha(u_{i,j}) = 0$ if and only if c_j is divisible by $p(i)$.

  - Constraints that ensure that exactly one of $\alpha(u_{i,1})$, $\alpha(u_{i,2})$, and $\alpha(u_{i,3})$ is equal to 0, i.e., the constraints $u_{i,1} + u_{i,2} + u_{i,3} = 2$. Note that together all the above constraints now ensure condition C1 holds.

- The following constraints, introduced for each $1 \leq j \leq 3$, together guarantee that condition C2 holds:
  - Constraints that ensure that for every $e \in E(G)$ and $i \in e$, it holds that $\alpha(r_{e,i,j})$ is equal to the remainder of c_j divided by $p(i)$, i.e., the constraint $c_j = p(i)m_{e,i,j} + r_{e,i,j}$.
  - Constraints that ensure that for every $e \in E(G)$, $i \in e$, and j with $1 \leq j \leq 3$ it holds that $\alpha(u_{e,i,j}) = 0$ if and only if $\alpha(r_{e,i,j}) = 0$, i.e., the constraints $u_{e,i,j} + \mathrm{sl}^3_{e,i,j} = r_{e,i,j}$ and $r_{e,i,j} + \mathrm{sl}^4_{e,i,j} = (p(i) - 1)u_{e,i,j}$. Note that together the above constraints now ensure that $\alpha(u_{e,i,j}) = 0$ if and only if c_j is divisible by $p(i)$.
  - Constraints that ensure that for every $e = \{i, k\} \in E(G)$ and j with $1 \leq j \leq 3$ it holds that at least one of $\alpha(u_{e,i,j})$ and $\alpha(u_{e,k,j})$ is non-zero, i.e., the constraint $u_{e,i,j} + u_{e,k,j} - \mathrm{sl}^5_{e,j} = 1$. Note that together with all of the above constraints this now ensures condition C2.

This completes the construction of $\mathbf{I}$. Clearly $\mathbf{I}$ can be constructed in polynomial time, and the largest coefficient used by $\mathbf{I}$ is equal to $p(|V(G)|)$. It is well-known that $p(i)$ is upper-bounded by $O(i \log i)$ due to the Prime Number Theorem, and so this in particular implies that the numbers which occur in $\mathbf{I}$ are bounded by a polynomial in $|V(G)|$.

Following the construction and explanations provided above, it is not difficult to see that $\mathbf{I}$ has a feasible assignment if and only if G has a 3-coloring. Indeed, for any 3-coloring of G, one can construct a feasible assignment of $\mathbf{I}$ by computing the prime-number encoding for vertices that receive colors $1, 2, 3$ and assign these three numbers to c_1, c_2, c_3, respectively. Such an assignment allows us to straightforwardly satisfy the constraints ensuring C1 holds (since each prime occurs in exactly one global constraint), the constraints ensuring C2 holds (since each edge is incident to at most one of each color) while maintaining the domain bounds.

On the other hand, for any feasible assignment α, clearly each of $\alpha(c_1), \alpha(c_2), \alpha(c_3)$ will be divisible by some subset of prime numbers between 2 and $p(|V(G)|)$. In particular, since α is feasible it follows from the construction of our first group of constraints that each prime between 2 and $p(|V(G)|)$ divides precisely one of $\alpha(c_1), \alpha(c_2), \alpha(c_3)$, and so this uniquely encodes a corresponding candidate 3-coloring for the vertices of the graph. Finally, since α also satisfies the second group of constraints, this candidate 3-coloring must have the property that each edge is incident to at exactly 2 colors, and so it is in fact a valid 3-coloring.

It remains to show that $\mathfrak{p}^V(\mathbf{I}) \leq 25$. We show this by showing that the set $B = \{c_1, c_2, c_3\}$ is a variable modulator set to 25-compactness. Note that the graph $G_{\mathbf{I}} \setminus \{c_1, c_2, c_3\}$ has only two types of components (all other components are isomorphic to one of the two types):

- for every i, $1 \leq i \leq |V(G)|$, one component containing the variables $m_{i,1}, m_{i,2}, m_{i,3}$, $\mathrm{sl}^1_{i,1}, \mathrm{sl}^1_{i,2}, \mathrm{sl}^1_{i,3}$, $\mathrm{sl}^2_{i,1}, \mathrm{sl}^2_{i,2}, \mathrm{sl}^2_{i,3}$, $r_{i,1}, r_{i,2}, r_{i,3}$, $u_{i,1}, u_{i,2}, u_{i,3}$. Moreover, these 15 variables occur in exactly 10 constraints together; these are the constraints introduced above to ensure condition C1. Hence the total size of these components is 25.
- for every $e = \{w, v\} \in E(G)$ and j with $1 \leq j \leq 3$, one component on the vertices $m_{e,w,j}$, $m_{e,v,j}$, $\mathrm{sl}^3_{e,w,j}$, $\mathrm{sl}^4_{e,v,j}$, $r_{e,w,j}$, $r_{e,v,j}$, $u_{e,w,j}$, $u_{e,v,j}$, and $\mathrm{sl}^5_{e,j}$. Moreover, these 9 variables occur in exactly 7 constraints together; these are the constraints introduced above to ensure condition C2. Hence the total size of these components is 16.

This show that B is a variable modulator to 25-compactness, as required. □

Theorem 9.24. Unary ILP-feasibility *is* W[1]*-hard parameterized by* $\mathfrak{p}^C(\mathbf{I})$.

Proof. We prove the theorem by a parameterized reduction from Multicolored Clique, which is well-known to be W[1]-complete [172]. Given an integer k and a k-partite graph G with partition $V_1, \dots, V_k$, the Multicolored Clique problem asks whether G contains a k-clique. In the following we denote by $E_{i,j}$ the set of all edges in G with one endpoint in V_i and the other endpoint in V_j, for every i and j with $1 \leq i < j \leq k$. To show the lemma, we will construct an instance $\mathbf{I}$ of ILP-feasibility in polynomial time that has a constraint modulator set of size $2k + 2\binom{k}{2}$ to 3-compactness and coefficients bounded by a polynomial in $|V(G)|$ such that G has a k-clique if and only if $\mathbf{I}$ has a feasible assignment.

The main idea behind the reduction is to first guess one vertex from each part V_i and one edge between every two parts V_i and V_j and to then verify that the selected vertices and edges form a k-clique in G.

The first step is achieved by introducing one binary variable for every vertex and edge of G together with $2k + 2\binom{k}{2}$ global constraints that ensure that (1) exactly one of the variables representing the vertices in V_i is set to one and (2) exactly one of the variables representing the edges between V_i and V_j is set to one. The second step, i.e., verifying that the chosen vertices and edges indeed form a k-clique of G, is achieved by identifying each vertex of G with a unique number such that the sum of any two numbers assigned to two vertices of G is unique. By identifying each edge of G with the sum of the numbers assigned to its endpoints, it is then possible to verify that the selected vertices and edges form a k-clique by checking whether the number assigned to the selected edge e is equal to the sum of the numbers assigned to the selected vertices in V_i and V_j. Sets of numbers for which the sum of every two numbers from the set is unique are also known as Sidon sequences. Indeed a Sidon sequence is a sequence of natural numbers such that the sum of every two distinct numbers in the sequence is unique. For our reduction we will need a Sidon sequence of $|V(G)|$ natural numbers, i.e., containing one number for each vertex of G. Since the numbers in the Sidon sequence will be used as coefficients of

I, we need to ensure that the largest of these numbers is bounded by a polynomial in G. Indeed [76] shows that a Sidon sequence containing n elements and whose largest element is at most $2p^2$, where p is the smallest prime number larger or equal to n can be constructed in polynomial time. Together with Bertrand's postulate [6], which states that for every natural number n there is a prime number between n and $2n$, we obtain that a Sidon sequence containing $|V(G)|$ numbers and whose largest element is at most $8|V(G)|^2$ can be found in polynomial time. In the following we will assume that we are given such a Sidon sequence $\mathcal{S}$ and we denote by $\mathcal{S}(i)$ the i-th element of $\mathcal{S}$ for any i with $1 \leq i \leq |V(G)|$. Moreover, we denote by $\max(\mathcal{S})$ and $\max_2(\mathcal{S})$ the largest element of $\mathcal{S}$ respectively the maximum sum of any two numbers in $\mathcal{S}$.

We are now ready to construct the instance **I** of ILP-feasibility such that G has a k-clique if and only if **I** has a feasible assignment. This instance **I** has the following variables:

- For every $v \in V(G)$ a binary variable v (with domain $\{0, 1\}$) that is 1 if v is selected to be in the k-clique and 0 otherwise.
- For every $e \in E(G)$ a binary variable e (with domain $\{0, 1\}$) that is 1 if e is selected to be in the k-clique and 0 otherwise.
- For every i with $1 \leq i \leq k$, a variable v_i (with unrestricted domain), which will be set to $\mathcal{S}(v)$ if the vertex $v \in V_i$ was selected to be in the k-clique.
- For every i and j with $1 \leq i < j \leq k$, a variable $e_{i,j}$ (with unrestricted domain), which will be set to $\mathcal{S}(v) + \mathcal{S}(u)$ if the edge $e \in E_{i,j}$ with $e = \{u, v\}$ was selected to be in the k-clique.

I has the following constraints:

- Constraints that restrict the domains of all variables as specified above, i.e.:
  - for every $v \in V(G)$, the constraints $0 \leq v \leq 1$.
  - for every $e \in E(G)$, the constraints $0 \leq e \leq 1$.

  We will denote by D the set of all these constraints.
- for every i with $1 \leq i \leq k$, the constraint $\sum_{v \in V_i} v = 1$, which ensures that from every part V_i exactly one vertex is selected to be in the k-clique. We will denote by V_{SEL} the set of all these constraints.
- for every i and j with $1 \leq i < j \leq k$, the constraint $\sum_{e \in E_{i,j}} e = 1$, which ensures that between any two parts V_i and V_j exactly one edge is selected to be in the k-clique. We will denote by E_{SEL} the set of all these constraints.

- for every i with $1 \leq i \leq k$, the constraint $\sum_{v \in V_i} \mathcal{S}(v)v = v_i$, which ensures that v_i is equal to $\mathcal{S}(v)$ whenever v is selected for the k-clique. We will denote by V_{ASS} the set of all these constraints.
- for every i and j with $1 \leq i < j \leq k$, the constraint $\sum_{e=\{u,v\} \in E_{i,j}}(\mathcal{S}(u) + \mathcal{S}(v))e = e_{i,j}$, which ensures that $e_{i,j}$ is equal to $\mathcal{S}(u) + \mathcal{S}(v)$ whenever the edge $e \in E_{i,j}$ with endpoints u and v is selected for the k-clique. We will denote by E_{ASS} the set of all these constraints.
- for every i and j with $1 \leq i < j \leq k$, the constraint $v_i + v_j = e_{i,j}$, which ensures that between any two parts V_i and V_j the vertices selected for the clique are equal to the endpoints of the edge chosen between the two parts. We will denote by VE_{CHECK} the set of all these constraints.

This completes the construction of **I**. Clearly **I** can be constructed in polynomial time, and the largest coefficient used by I is equal to $\max_2(\mathcal{S})$, which is at most $2\max(\mathcal{S}) \leq 16|V(G)|^2$. We first show that **I** has a small constraint modulator to 3-compactness, and hence our parameter can bounded in terms of k. Namely, we claim that the set $B = V_{SEL} \cup E_{SEL} \cup V_{ASS} \cup E_{ASS} \cup VE_{CHECK}$ of constraints of **I** is a constraint modulator of size at most $2k + 3\binom{k}{2}$ to 3-compactness. Clearly, the components of $G_{\mathbf{I}} \setminus B$ have size at most 3, i.e., $G_{\mathbf{I}}$ has one component of size one for every variable in $\{v_1, \ldots, v_k, e_{1,2}, \ldots, e_{k-1,k}\}$ as well as one component of size 3 for every $a \in V(G) \cup E(G)$, containing the variable a together with the two constraints $0 \leq a$ and $a \leq 1$. Moreover, the sets V_{SEL}, E_{SEL}, V_{ASS}, E_{ASS}, and VE_{CHECK} have sizes at most k, $\binom{k}{2}$, k, $\binom{k}{2}$, and $\binom{k}{2}$, respectively, which implies that $|B| \leq 2k + 3\binom{k}{2}$.

It remains to show that G has k-clique if and only if **I** is feasible. For the forward direction suppose that G has a k-clique on the vertices $c_1, \ldots, c_k$, where $c_i \in V_i$ for every i with $1 \leq i \leq k$. Then it is straightforward to verify that the assignment α with:

- $\alpha(c_i) = 1$ for every i with $1 \leq i \leq k$ and $\alpha(v) = 0$ for every $v \in V(G) \setminus \{c_1, \ldots, c_k\}$,
- $\alpha(\{c_i, c_j\}) = 1$ for every i and j with $1 \leq i < j \leq k$ and $\alpha(e) = 0$ for every $e \in E(G) \setminus \{\, \{c_i, c_j\} \mid 1 \leq i < j \leq k \,\}$,
- $\alpha(v_i) = \mathcal{S}(c_i)$ for every i with $1 \leq i \leq k$, and
- $\alpha(e_{i,j}) = \mathcal{S}(c_i) + \mathcal{S}(c_j)$ for every i and j with $1 \leq i < j \leq k$

is a feasible assignment for **I**.

For the reverse direction suppose that we are given a feasible assignment α for **I**. Then because α satisfies the constraints in $D \cup V_{SEL} \cup E_{SEL}$ we obtain that for every i and j with $1 \leq i < j \leq k$ it holds that exactly one of the variables in V_i and exactly one of the variables in $E_{i,j}$ is set to one. Let c_i denote the unique vertex in V_i with $\alpha(c_i) = 1$ and similarly let $d_{i,j}$ denote the unique edge in $E_{i,j}$ with $\alpha(d_{i,j}) = 1$. It follows from the constraints in V_{ASS} that $\alpha(v_i) = \mathcal{S}(c_i)$ and similarly using the constraints in E_{ASS} we obtain that $\alpha(e_{i,j}) = \mathcal{S}(u) + \mathcal{S}(v)$, where u and v are the endpoints of the edge $d_{i,j}$ in G. Moreover, we obtain from the constraints in VE_{CHECK} that $v_i + v_j = e_{i,j}$ and hence $\mathcal{S}(c_i) + \mathcal{S}(c_j) = \mathcal{S}(u) + \mathcal{S}(v)$, where again u and v are the endpoints of the edge $d_{i,j}$ in G. Because $\mathcal{S}$ is a Sidon sequence, it follows that this can only hold if $\mathcal{S}(c_i) = \mathcal{S}(u)$ and $\mathcal{S}(c_j) = \mathcal{S}(v)$, which implies that $c_i = u$ and $c_j = v$. This shows that the endpoints of the selected edges $d_{1,2}, \dots, d_{k-1,k}$ are the vertices in $c_1, \dots, c_k$ and hence $G[\{c_1, \dots, c_k\}]$ is a k-clique of G. □

We conclude with a simple reduction ruling out an extension of our XP algorithm to general ILPs parameterized by $\mathfrak{p}^c$.

Theorem 9.25. ILP *is* NP*-hard even if* $\mathfrak{p}^C = 1$.

Proof. We show the result by a polynomial reduction from the Subset Sum problem, which is well-known to be weakly NP-complete. Given a set $S := \{s_1, \dots, s_n\}$ of integers and an integer s, the Subset Sum problem asks whether there is a subset $S' \subseteq S$ such that $\sum_{s \in S'} s' = s$. Let $I := (S, s)$ with $S := \{s_1, \dots, s_n\}$ be an instance of Subset Sum. We will construct an equivalent ILP instance $\mathbf{I}$ with $\mathfrak{p}^C(\mathbf{I}) = 1$ in polynomial time as follows. The instance $\mathbf{I}$ has n binary variables $x_1, \dots, x_n$ and apart from the domain constraints for these variables only one global constraint defined by $\sum_{1 \leq i \leq n} s_i x_i = s$. Because $\mathbf{I}$ has only one constraint, it holds that $\mathfrak{p}^C(\mathbf{I}) = 1$ and moreover it is straightforward to verify that $\mathbf{I}$ is equivalent to (S, s) (this has also for instance been shown in [129, Theorem 1]). □

9.6 Summary and Open Questions

In order to overcome the complexity barriers of ILP, a wide range of problems have been encoded in restricted variants of ILP such as 2-stage stochastic ILP and N-fold ILP; examples for the former include a range of transportation and logistic problems [173, 120], while examples for the latter range from scheduling [142] to, e.g., computational social choice [143]. Our framework based on *fracture modulators* provides a unified platform which generalizes 2-stage stochastic ILP, N-fold ILP and also 4-block N-fold ILP. More importantly though, it represents a natural measure of the complexity of ILPs which can be applied to any ILP instance, including those which lie outside of the scope of all previously known algorithmic frameworks. In fact, one may view our algorithmic results as “algorithmic meta-theorems” for ILP, where previously known algorithms for 2-stage stochastic ILP, N-fold ILP and 4-block N-fold ILP only represent a simple base case.

Our algorithms are complemented with matching lower bounds showing that the considered restrictions are, in fact, necessary in order to obtain fixed-parameter or XP algorithms. The only remaining blank part in the presented complexity map is the question of whether mixed fracture modulators admit a fixed-parameter algorithm in case of bounded coefficients; we believe that this is in fact a major open problem in the area. A first step towards settling this question would be to resolve the fixed-parameter (in)tractability of 4-block N-fold ILP, which was also left open in previous work; progress in this direction seems to require new techniques and insights [116].

Notes

The results in this chapter appeared in a conference paper in the proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2017) [67].

Part III

Hybrid parameters

CHAPTER 10

Well-Structured Modulators

The primary goal in this part is to push the boundaries of tractability for a wide range of problems beyond the state of the art for both of the approaches introduced in Parts I and II. More precisely, we investigate what happens when a graph contains a modulator which is large but "well-structured" (in the sense of having bounded rank-width). Can such modulators still be exploited to obtain efficient algorithms? And is it even possible to find such modulators efficiently?

Organization of the Chapter

This Chapter is devoted to introducing our novel family of parameters that combine the approaches introduced in Parts I andII. We start in the Section 10.1 by introducing the well-structured modulators to some class of graphs $\mathcal{H}$ and showing that they are more general then both standard modulator to $\mathcal{H}$ and rank-width of the input graph. Next, in Sections 10.2 and 10.3, we focus on building some crucial machinery that helps us find optimal well-structured modulators in FPT time (Chapter 11) or good approximate well-structured modulators (Chapter 12).

10.1 (k, c)-Well-Structured Modulators

Before we can formalize well-structured modulators, we need to further introduce some necessary notions.

Recall that a modulator to a graph class $\mathcal{H}$ is a vertex-subset of a graph G such that its deletion puts G into $\mathcal{H}$. Furthermore, recall the definition of rank-width from the Section 3.3. A graph class is called hereditary if it is closed under vertex deletion. Let $\mathcal{F}$ be a finite set of graphs; then the class of *$\mathcal{F}$-free* graphs is the class of all graphs which do not contain any graph in $\mathcal{F}$ as an induced subgraph. We will often refer to elements of $\mathcal{F}$ as obstructions, and we say that the class of $\mathcal{F}$-free graphs is *characterized by $\mathcal{F}$*.

10.1.1 Splits and Split-Modules

The notions of split and split decomposition (recall the Section 7.2) play an important role in particular when formally defining our parameters and for computing them. Recall that a split of a connected graph $G = (V, E)$ is a vertex bipartition (A, B) of V such that every vertex of $A' = N(B)$ has the same neighborhood in $B' = N(A)$. Let $G = (V, E)$ be a graph. To simplify our exposition, we will use the notion of *split-modules* instead of splits where suitable. A set $A \subseteq V$ is called a *split-module* of G if there exists a connected component $G' = (V', E')$ of G such that $A \subseteq V'$ and $(A, V' \setminus A)$ forms a split of G'. Notice that if A is a split-module then A can be partitioned into A_1 and A_2 such that $N(A_2) \subseteq A$ and for each $v_1, v_2 \in A_1$ it holds that $N(v_1) \cap (V' \setminus A) = N(v_2) \cap (V' \setminus A)$. We call the set A_1 a frontier of the split-module A and we denote it by $\lambda(A)$. For technical reasons, $\emptyset$ as well as $V(G')$, $V(G') \setminus \{v\}$, and v, for all connected components G' of G and all vertices $v \in V(G')$, are also considered split-modules. We say that two disjoint split-modules $X, Y \subseteq V$ are *adjacent* if there exist $x \in X$ and $y \in Y$ such that x and y are adjacent.

We are now ready to define well-structured modulators.

Definition 10.1. Let $\mathcal{H}$ be a graph class and let G be a graph. A set $\vec{X}$ of pairwise-disjoint split-modules of G is called a (k, c)-well-structured modulator to $\mathcal{H}$ if

1. $|\vec{X}| \leq k$, and
2. $\bigcup_{X_i \in \vec{X}} X_i$ is a modulator to $\mathcal{H}$, and
3. for each $X_i \in \vec{X}$:
    - if $G[X_i]$ is a connected component of G, then $\mathrm{rw}(G[X_i]) \leq c + 1$;
    - if $G[X_i]$ is not a connected component of G, then $\mathrm{rw}(G[X_i]) \leq c$.

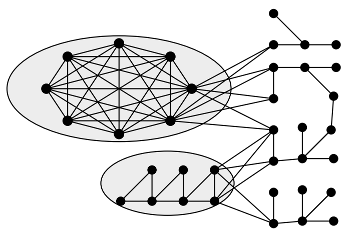


Figure 10.1: A graph with a $(2, 1)$-well-structured modulator to forests (in the two shaded areas).

For the sake of brevity and when clear from context, we will sometimes identify $\vec{X}$ with $\bigcup_{X_i \in \vec{X}} X_i$ (for instance $G - \vec{X}$ is shorthand for $G - \bigcup_{X_i \in \vec{X}} X_i$). To allow a concise description of our parameters, for any hereditary graph class $\mathcal{H}$ we let the c-well-structure number ($\mathrm{wsn}_c^{\mathcal{H}}$ in short) denote the minimum k such that G has a (k, c)-well-structured modulator to $\mathcal{H}$. Moreover, we let the well-structure number (wsn$^{\mathcal{H}}$ in short) denote the minimum k such that G has a (k, k)-well-structured modulator to $\mathcal{H}$. Moreover, we refer to (k, k)-well-structured modulator to $\mathcal{H}$ simply as k-well-structured modulator to $\mathcal{H}$. See Figure 10.1 for an illustration of a well-structured modulator.

Proposition 10.2. *Let $\mathcal{H}$ be an arbitrary hereditary graph class of unbounded rank-width.*

1. $\mathrm{rw}(G) \geq \mathrm{wsn}^{\mathcal{H}}(G)$ *for every graph G. Furthermore, for every $i \in \mathbb{N}$ there exists a graph G_i such that* $\mathrm{rw}(G_i) \geq \mathrm{wsn}^{\mathcal{H}}(G_i) + i$, *and*
2. *$mod^{\mathcal{H}}(G) \geq$* $\mathrm{wsn}^{\mathcal{H}}(G)$ *for every graph G. Furthermore, for every $i \in \mathbb{N}$ there exists a graph G_i such that $mod^{\mathcal{H}}(G_i) \geq$* $\mathrm{wsn}^{\mathcal{H}}(G_i) + i$.

Proof.

1. For $\mathrm{rw}(G) \geq \mathrm{wsn}^{\mathcal{H}}(G)$ notice that for every graph G of rank-width k, the set $\{V(G)\}$ is a k-well-structured modulator to the empty graph. For the second claim, since $\mathcal{H}$ has unbounded rank-width, for every $i \in \mathbb{N}$ it contains some graph G_i such that $\mathrm{rw}(G_i) > i$; by definition, $\mathrm{wsn}^{\mathcal{H}}(G_i) = 0$.
2. For $mod^{\mathcal{H}}(G) \geq \mathrm{wsn}^{\mathcal{H}}(G)$, let G be a graph containing a modulator $X = \{v_1, \ldots, v_k\}$ to $\mathcal{H}$. It is easy to check that $\vec{X} = \{\{v_1\}, \ldots, \{v_k\}\}$ is a k-well-structured modulator to $\mathcal{H}$. For the second claim, let $G' \notin \mathcal{H}$ and let $k = \mathrm{rw}(G')$. Consider the graph G_i consisting of $i + 1 + k$ disjoint copies of G' and a vertex q which is adjacent to every other vertex of G. Since $\mathcal{H}$ is hereditary, we may assume without loss of generality that it contains the single-vertex graph. It is then easy to check that $\{V(G) \setminus \{q\}\}$ forms a k-well-structured modulator in G to $\mathcal{H}$. Now consider an arbitrary set $X \subseteq V(G)$ of cardinality at most $i + k$. Clearly, there must exist some copy of G', say G'_j, such that $X \cap V(G'_j) = \emptyset$. Since $G'_j \notin \mathcal{H}$, it follows from the hereditarity of $\mathcal{H}$ that $G - X \notin \mathcal{H}$ and hence X cannot be a modulator to $\mathcal{H}$. We conclude $mod^{\mathcal{H}}(G_i) > i + k = i + \mathrm{wsn}^{\mathcal{H}}(G_i)$. □

Note that Proposition 10.2 implies that, since wsn$^{\mathcal{H}}$ lower-bounds rank-width and rank-width is known not to admit polynomial kernels for nearly any NP-hard problems [25], one cannot hope to use wsn$^{\mathcal{H}}$ for polynomial kernelization. The above is however not true if we consider $\mathrm{wsn}_c^{\mathcal{H}}$ for some fixed constant c. Therefore, we use k-well-structured modulators when developing FPT algorithms and (k, c)-well-structured modulators for developing polynomial kernels.

We conclude this section with a brief discussion on the choice of the parameter. The specific conditions restricting the contents of the modulator $\bigcup \vec{X}$ have been chosen as the most general means which allow both (1) the efficient finding of a suitable well-structured modulator, and (2) the efficient use of this well-structured modulator for FPT algorithm and kernelization. In this sense, we do not claim that there is anything inherently special about rank-width or split-modules, other than being the most general notions which are currently known to allow the achievement of these two goals.

10.2 Structural Properties of Split-Modules

The objective of this section is develop the main machinery that helps us develop various FPT and approximation algorithms for finding well-structured modulators in the following chapters.

Our first course of action is the statement of several useful properties of splits in graphs. We remark that for most of this section we will restrict ourselves to connected graphs, and show how to deal with general graphs later on; this allows us to use the following result by Cunningham.

Theorem 10.3 ([51]). *Let (A, C), (B, D) be splits of a connected graph G such that $|A \cap B| \geq 2$ and $A \cup B \neq V(G)$. Then $(A \cap B, C \cup D)$ is a split of G.*

Two vertex sets A, B are overlapping if $A \cap B$, $A \setminus B$, and $B \setminus A$ are all nonempty.

Lemma 10.4. *If A and B are overlapping split-modules of a connected graph $G = (V, E)$, then $A \cup B$ is also a split-module. Moreover, if $A \cup B \neq V$, then also $A \cap B$ is a split-module.*

Proof. If $V = A \cup B$, then $A \cup B$ is clearly a split-module. So, assume $A \cup B \neq V$ and let $C = V \setminus A$ and $D = V \setminus B$; note that $C \cup D \neq V$ since A, B are overlapping. We make the following exhaustive case distinction:

- if $|A \cap B| = 1$ and $|C \cap D| = 1$, then both $A \cap B$ and $A \cup B = V \setminus (C \cap D)$ are easily seen to be split-modules;
- if $|A \cap B| \geq 2$ and $|C \cap D| = 1$, then $A \cap B$ is a split-module by Theorem 10.3 and $A \cup B$ is also a split-module because $C \cap D$ is a split-module;
- if $|A \cap B| = 1$ and $|C \cap D| \geq 2$, then $A \cap B$ is a split-module and $A \cup B$ is also a split-module because C, D satisfy the conditions of Theorem 10.3 and hence $C \cap D = V \setminus (A \cup B)$ forms a split-module;
- if $|A \cap B| \geq 2$ and $|C \cap D| \geq 2$, then $A \cap B$ is a split-module by Theorem 10.3 and $A \cup B$ is also a split-module because C, D satisfy the conditions of Theorem 10.3, as in the previous case. ☐

Lemma 10.5. *Let $G = (V, E)$ be a connected graph and A, B be overlapping split-modules. Then $A \setminus B$ is also a split-module.*

Proof. The lemma clearly holds if $|A \setminus B| \leq 1$, so we may assume that $|A \setminus B| \geq 2$. Let $Z = V \setminus B$; since B is a split module, so is Z. Furthermore, since A and B are overlapping, it holds that $B \setminus A$ is nonempty and hence $V \neq Z \cup A$. Since $Z \cap A = A \setminus B$, we have $|Z \cap A| \geq 2$ and hence we conclude that $Z \cap A = A \setminus B$ is a split module by Theorem 10.3. ☐

Lemma 10.6. *Let $k \in \mathbb{N}$, $G = (V, E)$ be a graph, and A, B, C be pairwise disjoint split-modules such that $A \cup B \cup C = V$. Let a, b, c be arbitrary vertices such that $a \in N(A)$, $b \in N(B)$, and $c \in N(C)$. If* $\max\left(\mathrm{rw}(G[A \cup \{a\}]), \mathrm{rw}(G[B \cup \{b\}]), \mathrm{rw}(G[C \cup \{c\}])\right) \leq k$, *then* $\mathrm{rw}(G) \leq k$.

Proof. Let $\mathcal{T}_A = (T_A, \mu_A)$, $\mathcal{T}_B = (T_B, \mu_B)$, and $\mathcal{T}_C = (T_C, \mu_C)$ be witnessing rank decompositions of $G[A], G[B]$, and $G[C]$, respectively.

We construct a rank decomposition $\mathcal{T} = (T, \mu)$ of G as follows.

Let l_a be the leaf (note that μ_A is bijective) of T_A such that $\mu_A(a) = l_a$. Similarly, let l_b and l_c be the leaves such that $\mu_B(b) = l_b$ and $\mu_C(c) = l_c$, respectively. We obtain T from T_A by adding disjoint copies of T_B and T_C and then identifying l_a with the copies of l_b and l_c. Since T_A, T_B, and T_C are subcubic, so is T.

We define the mapping $\mu : V(G) \to \{\, t \mid \text{ t is a leaf of } T \,\}$ by

$$\mu(v) = \begin{cases} \mu_a(v) & \text{if } v \in A, \\ g(\mu_b(v)) & \text{if } v \in B, \\ g(\mu_c(v)) & \text{otherwise,} \end{cases}$$

where g maps internal nodes in $T_B \cup T_C$ to their copies in T. The mappings μ_A, μ_B, and μ_C are bijections and g is injective, so μ is injective. By construction, the image of $V(G)$ under μ is the set of leaves of T, so μ is a bijection. Thus $\mathcal{T} = (T, \mu)$ is a rank decomposition of G.

We prove that the width of $\mathcal{T}$ is at most k. Given a rank decomposition $\mathcal{T}^* = (T^*, \mu^*)$ and an edge e of T^*, the connected components of $T^* - e$ induce a bipartition (X, Y) of the leaves of T^*. We set $f : (\mathcal{T}^*, e) \mapsto ({\mu^*}^{-1}(X), {\mu^*}^{-1}(Y))$. Take any edge e of T. There is a natural bijection β from the edges in T to the edges of $T_A \cup T_B \cup T_C$. Accordingly, we distinguish three cases for $e' = \beta(e)$:

1. $e' \in T_A$. Let $(U, W) = f(\mathcal{T}_A, e')$. Without loss of generality assume that $a \in W$. Then by construction of $\mathcal{T}$, we have $f(\mathcal{T}, e) = (U, W \cup B \cup C)$. Let $u \in A$ and $v \in B \cup C$. Since A is split-module either $v \notin N(A)$ and $\mathbf{A}_G(u, v) = 0$ for all $u \in A$, or $v \in N(A)$ in which case $\mathbf{A}_G(u, v) = \mathbf{A}_G(u, a)$ for all $u \in A$. Therefore, to obtain $\mathbf{A}_G(U, W \cup B \cup C)$ one can simply copy the column corresponding to a in $\mathbf{A}_G(U, W)$ or add some empty columns. This does not increase the rank of the matrix.

2. $e' \in T_B$. This case is symmetric to case 1, with A and B switching their roles and b taking the role of a.

3. $e' \in T_C$. This case is symmetric to case 1, with A and C switching their roles and c taking the role of a.

Since β is bijective, this proves that the rank of any bipartite adjacency matrix induced by removing an edge $e \in T$ is bounded by k. We conclude that the width of $\mathcal{T}$ is at most k and thus $\mathrm{rw}(G) \leq k$. □

By repeating the proof technique of Lemma 10.6 without the set C, we obtain the following corollary.

Corollary 10.7. *Let $k \in \mathbb{N}$, $G = (V, E)$ be a graph, and A, B pairwise disjoint split-modules such that $A \cup B = V$. Let $a, b \in V$ be such that $a \in N(A)$ and $b \in N(B)$. If $\max\left(\mathrm{rw}(G[A \cup \{a\}]), \mathrm{rw}(G[B \cup \{b\}])\right) \leq k$, then $\mathrm{rw}(G) \leq k$.*

Lemma 10.8. *Let $k \in \mathbb{N}$ and let $G = (V, E)$ be a connected graph having split-modules M_1, M_2 where $M_1 \cup M_2 = V$ and $\max(\mathrm{rw}(G[M_1]), \mathrm{rw}(G[M_2])) \leq k$. Then $\mathrm{rw}(G) \leq k+1$.*

Proof. Let $M_{22} = M_2 \setminus M_1$. Clearly, (M_1, M_{22}) is a split. Since rank-width is preserved by taking induced subgraphs, the graph $G[M_{22}]$ has rank-width at most k. Let $v_1 \in N(M_{22})$ and $v_2 \in N(M_1)$. It is easy to see that graphs $G_1 = G[M_1 \cup \{v_2\}]$ and $G_2 = G[M_{22} \cup \{v_1\}]$ have rank-width at most $k + 1$. We finish the proof by applying Corollary 10.7, using M_1, M_{22} in roles of A, B and v_1, v_2 in roles of a, b, respectively. □

The following lemma in essence shows that the relation of being in a split-module of small rank-width is transitive (assuming sufficiently high rank-width). The significance of this will become clear later on.

Lemma 10.9. *Let $k \in \mathbb{N}$. Let $G = (V, E)$ be a connected graph of rank-width at least $k + 2$ and let M_1, M_2 be split-modules of G such that $M_1 \cap M_2 \neq \emptyset$ and it holds that $\max(\mathrm{rw}(G[M_1]), \mathrm{rw}(G[M_2])) \leq k$. Then $M_1 \cup M_2$ is a split-module of G and $\mathrm{rw}(G[M_1 \cup M_2]) \leq k$.*

Proof. If $M_1 \subseteq M_2$ or $M_2 \subseteq M_1$ the result is immediate, hence we may assume that they are overlapping. Lemma 10.8 and $\mathrm{rw}(G) \geq k + 2$ together imply that $M_1 \cup M_2 \neq V$. Let $M_{11} = M_1 \setminus M_2, M_{22} = M_2 \setminus M_1$, and $M_{12} = M_1 \cap M_2$. It follows from Lemma 10.4 and Lemma 10.5 that these sets are split-modules of G. Let $v_{11} \in N(V \setminus M_{11}), v_{22} \in N(V \setminus M_{22})$, and $v_{12} \in N(V \setminus M_{12})$. We show that $\mathrm{rw}(G[M_1 \cup M_2]) \leq k$. By assumption, both $G[M_1]$ and $G[M_2]$ have rank-width at most k. Since rank-width is preserved by taking induced subgraphs, the graphs $G_{11} = G[M_{11} \cup \{v_{12}\}]$, $G_{12} = G[M_{12} \cup \{v_{22}\}]$, and $G_{22} = G[M_{22} \cup \{v_{12}\}]$ also have rank-width at most k. We finish the proof by applying Lemma 10.6, where M_{11}, M_{22}, M_{12} take the roles of A, B, and C and v_{12}, v_{12}, and v_{22} take the roles of a, b, and c, respectively. □

Definition 10.10. Let $k \in \mathbb{N}$ and let G be a graph. We define a relation $\sim_k^G$ on $V(G)$ by letting $v \sim_k^G w$ if and only if at least one of the following conditions holds:

- there is a connected component G' of G such that $v, w \in V(G')$ and $\mathrm{rw}(G') \leq k+1$,

- there is a split-module M of G such that $v, w \in M$ and $\mathrm{rw}(G[M]) \leq k$.

We drop the superscript from $\sim_k^G$ if the graph G is clear from context.

If $\sim$ is an equivalence relation over a set A, then for $a \in A$ we use $[a]_\sim$ to denote the equivalence class containing a. Using Lemma 10.9 to deal with transitivity, we prove the following.

Proposition 10.11. *For every $k \in \mathbb{N}$ and a connected graph $G = (V, E)$, the relation $\sim_k$ is an equivalence relation, and each equivalence class U of $\sim_k$ is a split-module of G such that $\mathrm{rw}(G[U]) \leq k + 1$.*

Proof. If $\mathrm{rw}(G) \leq k + 1$, then $\sim_k$ contains exactly one equivalence class containing all of the vertices of G and the proposition trivially holds. Therefore, for the rest of the proof assume that $\mathrm{rw}(G) \geq k + 2$.

For every $v \in V$, the singleton $\{v\}$ is a split-module of G, so $\sim_k$ is reflexive. Symmetry of $\sim_k$ is trivial. For transitivity, let $u, v, w \in V$ be such that $u \sim_k v$ and $v \sim_k w$. Then there are split-modules M_1, M_2 of G such that $u, v \in M_1$, $v, w \in M_2$, and $\mathrm{rw}(G[M_1]), \mathrm{rw}(G[M_2]) \leq k$.By Lemma 10.9, $M_1 \cup M_2$ is a split-module of G such that $\mathrm{rw}(G[M_1 \cup M_2]) \leq k$. In combination with $u, w \in M_1 \cup M_2$ that implies $u \sim_k w$. This concludes the proof that $\sim_k$ is an equivalence relation.

Now let $v \in V$ and let $U = [v]_{\sim_k}$. For each $u \in U$ there is a split-module W_u of G such that $u, v \in W_u$ and $\mathrm{rw}(G[W_u]) \leq k$. By Lemma 10.9, $W = \bigcup_{u \in U} W_u$ is a split-module of G and $\mathrm{rw}(G[W]) \leq k$. Clearly, $[v]_{\sim_k} \subseteq W$. On the other hand, $u \in W$ implies $v \sim_k u$ by definition of $\sim_k$, so $W \subseteq [v]_{\sim_k}$. That is, $W = [v]_{\sim_k}$. □

Since every split-module of a graph G is a subset of some connected component of G, it is easy to see the following simple but useful observation.

Observation 10.12. *Let $k \in \mathbb{N}$, G be a disconnected graph, and $\mathcal{C}(G)$ be the set of connected components of G. Then $\sim_k^G = \bigcup_{G' \in \mathcal{C}(G)} \sim_k^{G'}$.*

From Observation 10.12 and Proposition 10.11 it immediately follows that $\sim_k$ is an equivalence relation, and each equivalence class U of $\sim_k$ is a split-module of G such that $\mathrm{rw}(G[U]) \leq k + 1$ holds even for disconnected graphs.

Corollary 10.13. *Every graph G has its vertex set uniquely partitioned by the equivalence classes of $\sim_k$ into inclusion-maximal split-modules of rank-width at most k and connected components of G or rank-width at most $k + 1$.*

Now that we know $\sim_k$ is an equivalence relation, we show how to compute it in FPT time. Before we give an FPT algorithm for computing $\sim_k$, we need to state the following useful result.

Theorem 10.14 ([51, 104, 105, 106])**.** *Let D be the canonical split decomposition of a connected graph G. Every split of G is the bipartition (of unmarked vertices) induced by removing an marked edge from D', where $D' = D$ or D' is obtained from D by exactly one simple decomposition of a degenerate bag (i.e., a bag that is a clique or a star).*

Furthermore, it will be useful to recall Theorem 7.7 from Section 7.2.

Proposition 10.15. *Let $k \in \mathbb{N}$. Given an n-vertex graph G and two vertices v, w, we can decide whether $v \sim_k w$ in time $f(k) \cdot n^3$ for some computable function f.*

Proof. From Observation 10.12 it follows that if the proposition holds for connected graphs, then it holds for disconnected graphs as well. Moreover, if $\mathrm{rw}(G) \leq k+1$, it follows from the definition of the equivalence relation $\sim_k$ that $v \sim_k w$ for all pairs of vertices in G. Hence, we may assume that G is connected and $\mathrm{rw}(G) \geq k+2$.

By Theorem 7.7 we can compute the unique canonical split decomposition D in $\mathcal{O}(|V(G)| + |E(G)|)$ time. Due to Theorem 10.14, every split in G is the bipartition of unmarked vertices induced by removing an marked edge from D', where $D' = D$ or D' is obtained from D by exactly one simple decomposition of a degenerate bag.

All vertices of G are precisely the unmarked vertices of D and we can find a path P between the bag B_1 containing v and the bag B_2 containing w in D in time linear in the number of bags of D. Since the number of bags in a split decomposition is linear in the number of vertices of the original graph [102], there are at most linearly many bags on the path. Moreover, if B is a degenerate node on P, then we can split it, by computing its simple decomposition, into two bags B_1, B_2 in a way such that B_1 contains exactly the two marked vertices of B incident to a marked edge of D on P and a new marker vertex connecting it to B_2. We split all degenerate bags on P in this way and denote the marked graph by D'. Note that now every degenerate bag on a new path P' between u and v has 3 vertices.

Now every edge between P' and $D' \setminus P'$ corresponds to a minimal split-module containing v and w. Conversely, as a consequence of Theorem 10.14 every minimal split-module containing v and w is induced by removing a marked edge between P' and $D' \setminus P'$, and let M_{vw} be the set containing all of these at most $|D|$ minimal split modules. Hence, $v \sim_k w$ if and only if there is a split-module X in M_{vw} such that $\mathrm{rw}(G[X]) \leq k$. By Theorem 3.8 we can decide, for each such X, whether $\mathrm{rw}(G[X]) \leq k$ in time $f(k) \cdot n^3$, where f is some computable function. □

10.3 MSO Machinery for Well-Structured Modulators

We first review MSO-*types* roughly following the presentation in the textbook by Libkin [148]. The quantifier rank of an MSO formula φ is defined as the nesting depth of quantifiers in φ. For non-negative integers q and l, let $\mathsf{MSO}_{q,l}$ consist of all MSO formulas of quantifier rank at most q having at most l free set variables.

Let $\varphi = \varphi(X_1, \dots, X_l)$ and $\psi = \psi(X_1, \dots, X_l)$ be MSO formulas. We say φ and ψ are *equivalent*, written $\varphi \equiv \psi$, if for all graphs G and $U_1, \dots, U_l \subseteq V(G)$, $G \models \varphi(U_1, \dots, U_l)$ if and only if $G \models \psi(U_1, \dots, U_l)$. Given a set F of formulas, let $F/{\equiv}$ denote the set of equivalence classes of F with respect to $\equiv$. A system of representatives of $F/{\equiv}$ is a set $R \subseteq F$ such that $R \cap C \neq \emptyset$ for each equivalence class $C \in F/{\equiv}$. The following statement has a straightforward proof using normal forms (see [148, Proposition 7.5 and Lemma 3.13] for details).

Fact 10.16 ([97]). *Let q and l be fixed non-negative integers. The set $\mathsf{MSO}_{q,l}/{\equiv}$ is finite, and one can compute a finite system of representatives of $\mathsf{MSO}_{q,l}/{\equiv}$.*

Note that the system of representatives obtained in this way need not be inclusion-minimal, and we do not assume to have a mapping from this system of representatives to elements of $\mathsf{MSO}_{q,l}/{\equiv}$. We will assume that for every pair of non-negative integers q and l the system of representatives of $\mathsf{MSO}_{q,l}/{\equiv}$ given by Fact 10.16 is fixed.

Definition 10.17 (MSO Type). Let q, l be non-negative integers. For a graph G and an l-tuple $\vec{U}$ of sets of vertices of G, we define MSO-$type_q(G, \vec{U})$ as the set of formulas $\varphi \in \mathsf{MSO}_{q,l}$ such that $G \models \varphi(\vec{U})$. We call MSO-$type_q(G, \vec{U})$ the MSO *q-type of $\vec{U}$ in G.*

Since we will only be dealing with MSO logic, throughout the Part III we will refer to MSO-types simply as *types*. It follows from Fact 10.16 that up to logical equivalence, every type contains only finitely many formulas. This allows us to represent types using MSO formulas, as formalized in the next lemma. We remark that the statement of the next lemma used in previous work [97] did not specify the ("fixed-parameter") dependence of the running time on the rank-width, and so here we give a proof of the lemma for completeness.

Lemma 10.18 ([97]). *Let q and l be non-negative integer constants. Let G be a graph, and let $\vec{U}$ be an l-tuple of sets of vertices of G. One can compute a formula $\Phi \in \mathsf{MSO}_{q,l}$ such that for any graph G' and any l-tuple $\vec{U}'$ of sets of vertices of G' we have $G' \models \Phi(\vec{U}')$ if and only if $type_q(G, \vec{U}) = type_q(G', \vec{U}')$. Moreover, Φ can be computed in time $f(\mathrm{rw}(G)) \cdot |V|^3$.*

Proof. Let R be a system of representatives of $\mathsf{MSO}_{q,l}/{\equiv}$ given by Fact 10.16. Because q and l are constant, we can consider both the cardinality of R and the time required to compute it as constants. Let $\Phi \in \mathsf{MSO}_{q,l}$ be the formula defined as $\Phi = \bigwedge_{\varphi \in S} \varphi \wedge \bigwedge_{\varphi \in R \setminus S} \neg\varphi$, where $S = \{\, \varphi \in R \mid G \models \varphi(\vec{U}) \,\}$. We can compute Φ by deciding $G \models \varphi(\vec{U})$ for each $\varphi \in R$. Since the number of formulas in R is a constant, this can be done in time $f(rw(G)) \cdot |V|^3$ (for some computable function f) as checking whether G satisfied $\varphi(\vec{U})$ can be done in time $q(\mathrm{rw}(G)) \cdot |V|^3$ (for some computable function q).

Let G' be an arbitrary graph and let $\vec{U}'$ be an l-tuple of subsets of $V(G')$. We claim that $type_q(G, \vec{U}) = type_q(G', \vec{U}')$ if and only if $G' \models \Phi(\vec{U}')$. Since $\Phi \in \mathsf{MSO}_{q,l}$ the forward direction is trivial. For the converse, assume $type_q(G, \vec{U}) \neq type_q(G', \vec{U}')$. First suppose

$\varphi \in \mathit{type}_q(G,\vec{U}) \setminus \mathit{type}_q(G',\vec{U'})$. The set R is a system of representatives of $\mathsf{MSO}_{q,l}/\equiv$, so there has to be a $\psi \in R$ such that $\psi \equiv \varphi$. But $G' \models \Phi(\vec{U'})$ implies $G' \models \psi(\vec{U'})$ by construction of Φ and thus $G' \models \varphi(\vec{U'})$, a contradiction. Now suppose $\varphi \in \mathit{type}_q(G',\vec{U'}) \setminus \mathit{type}_q(G,\vec{U})$. An analogous argument proves that there has to be a $\psi \in R$ such that $\psi \equiv \varphi$ and $G' \models \neg\psi(\vec{U'})$. It follows that $G' \not\models \varphi(\vec{U'})$, which again yields a contradiction. □

The remainder of the section introduces the classical notion of MSO games (Definition 10.20) and their relation to MSO types (Theorem 10.21). However, to formally define MSO games, we first need the notion of partial isomorphism.

Definition 10.19 (Partial isomorphism)**.** Let G, G' be graphs, and let $\vec{V} = (V_1,\dots,V_l)$ and $\vec{U} = (U_1,\dots,U_l)$ be tuples of sets of vertices such that $V_i \subseteq V(G)$ and $U_i \subseteq V(G')$ for each $i \in [l]$. Let $\vec{v} = (v_1,\dots,v_m)$ and $\vec{u} = (u_1,\dots,u_m)$ be tuples of vertices such that $v_i \in V(G)$ and $u_i \in V(G')$ for each $i \in [m]$. Then $(\vec{v},\vec{u})$ defines a *partial isomorphism between* $(G,\vec{V})$ *and* $(G',\vec{U})$ if the following two conditions hold:

- For every $i, j \in [m]$,
$$v_i = v_j \;\Leftrightarrow\; u_i = u_j \text{ and } v_iv_j \in E(G) \;\Leftrightarrow\; u_iu_j \in E(G').$$
- For every $i \in [m]$ and $j \in [l]$,
$$v_i \in V_j \;\Leftrightarrow u_i \in U_j.$$

In the definition of MSO games given below, we denote the concatenation of tuple $\vec{A}$ by tuple $\vec{B}$ as $\vec{A} \frown \vec{B}$.

Definition 10.20 ([148], Definition 7.6)**.** Let G and G' be graphs, and let $\vec{V_0}$ be a k-tuple of subsets of $V(G)$ and let $\vec{U_0}$ be a k-tuple of subsets of $V(G')$. Let q be a non-negative integer. The *q-round* MSO *game on G and G' starting from* $(\vec{V_0},\vec{U_0})$ is played as follows. The game proceeds in rounds, and each round consists of one of the following kinds of moves.

Point move: The Spoiler picks a vertex in either G or G'; the Duplicator responds by picking a vertex in the other graph.

Set move: The Spoiler picks a subset of $V(G)$ or a subset of $V(G')$; the Duplicator responds by picking a subset of the vertex set of the other graph.

Let $\vec{v} = (v_1,\dots,v_m), v_i \in V(G)$ and $\vec{u} = (u_1,\dots,u_m), u_i \in V(G')$ be the point moves played in the q-round game, and let $\vec{V} = (V_1,\dots,V_l), V_i \subseteq V(G)$ and $\vec{U} = (U_1,\dots,U_l), U_i \subseteq V(G')$ be the set moves played in the q-round game, so that $l + m = q$ and moves belonging to same round have the same index. Then the Duplicator wins the game if $(\vec{v},\vec{u})$ is a partial isomorphism of $(G,\vec{V_0} \frown \vec{V})$ and $(G',\vec{U_0} \frown \vec{U})$. If the Duplicator has a winning strategy, we write $(G,\vec{V_0}) \equiv_q^{\mathsf{MSO}} (G',\vec{U_0})$.

Theorem 10.21 ([148], Theorem 7.7)**.** *Given two graphs G and G' and two l-tuples $\vec{V_0}, \vec{U_0}$ of sets of vertices of G and G', respectively, we have*

$$\mathit{type}_q(G, \vec{V_0}) = \mathit{type}_q(G', \vec{U_0}) \Leftrightarrow (G, \vec{V_0}) \equiv_q^{\mathsf{MSO}} (G', \vec{U_0}).$$

Our strategy for proving our most general results for an arbitrary fixed MSO_1 formula in the following chapters relies on a replacement technique, where each split-module in the well-structured modulator is replaced by a small representative. We use the notion of similarity defined below to prove that this procedure does not change the outcome of MSO-MC$_\varphi$.

Definition 10.22 (Similarity)**.** Let q and k be non-negative integers, $\mathcal{H}$ be a graph class, and let G and G' be graphs having k-well-structured modulators $\vec{X} = \{X_1, \ldots, X_k\}$ and $\vec{X'} = \{X'_1, \ldots, X'_k\}$ to $\mathcal{H}$, respectively. For $1 \leq i \leq k$, let S_i contain the frontier of split module X_i and similarly let S'_i contain the frontier of split module X'_i. We say that $(G, \vec{X})$ and $(G', \vec{X'})$ are *q-similar* if all of the following conditions are met:

1. There exists an isomorphism τ between $G - \vec{X}$ and $G' - \vec{X'}$.
2. For every $v \in V(G) \setminus \vec{X}$ and $i \in [k]$, it holds that v is adjacent to S_i if and only if $\tau(v)$ is adjacent to S'_i.
3. If $k \geq 2$, then for every $1 \leq i < j \leq k$ it holds that every $s_i \in S_i$ is adjacent to every $s_j \in S_j$ if and only if every $s'_i \in S'_i$ is adjacent to every $s'_j \in S'_j$.
4. For each $i \in [k]$, it holds that $\mathit{type}_q(G[X_i], S_i) = \mathit{type}_q(G'[X'_i], S'_i)$.

Lemma 10.23. *Let q, k, and c be non-negative integers, $\mathcal{H}$ be a graph class, and let G and G' be graphs having (k, c)-well-structured modulators $\vec{X} = \{X_1, \ldots, X_k\}$ and $\vec{X'} = \{X'_1, \ldots, X'_k\}$ to $\mathcal{H}$, respectively. If $(G, \vec{X})$ and $(G', \vec{X'})$ are q-*similar*, then it holds that $\mathit{type}_q(G, \emptyset) = \mathit{type}_q(G', \emptyset)$.*

Proof. For $i \in [k]$, we write $G_i = G[X_i]$ and $G'_i = G'[X'_i]$. Let $X_0 = V(G) \setminus \vec{X}$ and $X'_0 = V(G') \setminus \vec{X'}$. By Theorem 10.21, Condition 4 of Definition 10.22 is equivalent to $(G_i, S_i) \equiv_q^{\mathsf{MSO}} (G'_i, S'_i)$. That is, for each $i \in [k]$, Duplicator has a winning strategy π_i in the q-round MSO game played on G_i and G'_i starting from (S_i, S'_i). We construct a strategy witnessing $(G, \emptyset) \equiv_q^{\mathsf{MSO}} (G', \emptyset)$ in the following way:

1. Suppose Spoiler makes a set move W and assume without loss of generality that $W \subseteq V(G)$. For $i \in [k]$, let $W_i = X_i \cap W$, and let W'_i be Duplicator's response to W_i according to π_i. Furthermore, let $W'_0 = \{ \tau(v) \mid v \in W \cap X_0 \}$. Then Duplicator responds with $W' = W'_0 \cup \bigcup_{i=1}^{k} W'_i$.

2. Suppose Spoiler makes a point move s and again assume without loss of generality that $s \in V(G)$. If $s \in X_i$ for some $i \in [k]$, then Duplicator responds with $s' \in X'_i$ according to π_i; otherwise, Duplicator responds with $\tau(s)$ as per Definition 10.22 item 1.

Assume Duplicator plays according to this strategy and consider a play of the q-round MSO game on G and G' starting from $(\emptyset, \emptyset)$. Let $\vec{v} = (v_1, \ldots, v_m)$ and $\vec{u} = (u_1, \ldots, u_m)$ be the point moves in $V(G)$ and $V(G')$, respectively, and let $\vec{V} = (V_1, \ldots, V_l)$ and $\vec{U} = (U_1, \ldots, U_l)$ be the set moves in $V(G)$ and $V(G')$, respectively, so that $l + m = q$ and the moves made in the same round have the same index. We claim that $(\vec{v}, \vec{u})$ defines a partial isomorphism between $(G, \vec{V})$ and $(G', \vec{U})$.

- Let $j_1, j_2 \in [m]$ and let $v_{j_1}, v_{j_2} \in X_0$. Since τ is an isomorphism as per Definition 10.22 item 1, it follows that $v_{j_1} = v_{j_2}$ if and only if $u_{j_1} = u_{j_2}$ and $v_{j_1}v_{j_2} \in E(G)$ if and only if $u_{j_1}u_{j_2} \in E(G')$.

- Let $j_1, j_2 \in [m]$ and let $i \in [k]$ be such that $v_{j_1} \in X_0$ and $v_{j_2} \in X_i$. Then clearly $v_{j_1} \neq v_{j_2}$ and $u_{j_1} \neq u_{j_2}$. Consider the case $v_{j_1}v_{j_2} \in E(G)$. Then v_{j_2} must lie in the frontier of X_i, and hence $v_{j_2} \in S_i$. Since Duplicator's strategy π_i is winning for (G_i, S_i) and (G'_i, S'_i), it must hold that $u_{j_2} \in S'_i$. By Definition 10.22 item 2, it then follows that $\tau(v_{j_1})u_{j_2} \in E(G')$. So, consider the case $v_{j_1}v_{j_2} \notin E(G)$. Then either $v_{j_2} \notin S_i$, in which case it holds that $u_{j_2} \notin S'_i$ because of the choice of π_i and hence there cannot be an edge $u_{j_2}u_{j_1}$ in G', or $v_{j_2} \in S_i$, in which case it holds once again that $u_{j_2}u_{j_1} \notin E(G')$ by Definition 10.22 item 2.

- Let $j_1, j_2 \in [m]$ and let $i \in [k]$ be such that $v_{j_1}, v_{j_2} \in X_i$. Since Duplicator plays according to a winning strategy π_i in the game on G_i and G'_i, the restriction $(\vec{v}|_i, \vec{u}|_i)$ defines a partial isomorphism between $(G_i, (\vec{V})|_i)$ and $(G'_i, (\vec{U})|_i)$. It follows that $(v_{j_1}, v_{j_2}) \in E(G)$ if and only if $(u_{j_1}, u_{j_2}) \in E(G')$ and $v_{j_1} = v_{j_2}$ if and only if $u_{j_1} = u_{j_2}$.

- Let $j_1, j_2 \in [m]$ and let $i_1, i_2 \in [k]$ be pairwise distinct numbers such that $v_{j_1} \in X_{i_1}$ and $v_{j_2} \in X_{i_2}$. Then $v_{j_1} \neq v_{j_2}$ and also $u_{j_1} \neq u_{j_2}$ since $u_{j_1} \in X'_{i_1}$ and $u_{j_2} \in X'_{i_2}$ by the Duplicator's strategy. Suppose $v_{j_1}v_{j_2} \in E(G)$. Then $v_{j_1} \in S_{i_1}$, and $v_{j_2} \in S_{i_2}$, and S_{i_1} and S_{i_2} are adjacent in G. From the correctness of π_{i_1} and π_{i_2} it follows that $u_{j_1} \in S'_{i_1}$ and $u_{j_2} \in S'_{i_2}$, and from Definition 10.22 item 3 it follows that S'_{i_1} and S'_{i_2} are adjacent in G', which together implies $u_{j_1}u_{j_2} \in E(G')$. On the other hand, suppose $v_{j_1}v_{j_2} \notin E(G)$. Then either $v_{j_1} \notin S_{i_1}$, or $v_{j_2} \notin S_{i_2}$, or S_{i_1} and S_{i_2} are not adjacent in G. In the first case we have $u_{j_1} \notin S'_{i_1}$, in the second case we have $u_{j_2} \notin S'_{i_2}$, and in the third case it holds that S'_1 and S'_2 are not adjacent in G'; any of these three cases imply $u_{j_1}u_{j_2} \notin E(G')$.

- Let $j \in [m]$ such that $v_j \in X_0$. Then by the Duplicator's strategy on X_0 it follows that for any V_q such that $v_j \in V_q$ it holds that $u_j \in U_q$ and for any V_q such that $v_j \notin V_q$ it holds that $u_j \notin U_q$.

- Let $j \in [m]$ and $i \in [k]$ such that $v_j \in X_k$. Let V_q be such that $v_j \in V_q$. Since π_i is a winning strategy for Duplicator, it must be the case that $u_j \in U_q$. Similarly, if $v_j \notin V_q$ then the correctness of π_i guarantetes that $u_j \notin U_q$. □

Next, we show that small representatives can be computed efficiently.

Lemma 10.24. *Let $q \in \mathbb{N}_0$. There exist functions f, g such that one can compute, for an input graph G of rank-width at most k and $S \subseteq V(G)$, in time $f(k) \cdot |V(G)|^{\mathcal{O}(1)}$ a graph G' and a set $S' \subseteq V(G')$ such that $|V(G')| \leq g(q)$ and $type_q(G, S) = type_q(G', S')$.*

Proof. By Lemma 10.18 we can compute a formula $\Phi(Q)$ capturing the type T of (G, S) in time $f(k) \cdot |V(G)|^{\mathcal{O}(1)}$. Given $\Phi(Q)$, a constant-size model (G', S') satisfying $\Phi(Q)$ can be computed as follows. We start enumerating all graphs (by brute force and in any order with a non-decreasing number of vertices), and check for each graph G^* and every vertex-subset $S^* \subseteq V(G^*)$ whether $G^* \models \Phi(S^*)$. If this is the case, we stop and output (G^*, S^*). Since $G \models \Phi(S)$ this procedure must terminate eventually. Fixing the order in which graphs are enumerated, the number of graphs we have to check depends only on T. By Fact 10.16 the number of q-types is finite for each q, so we can think of the total number of checks and the size of each checked graph G^* as bounded by a constant. Moreover the time spent on each check depends only on T and the size of the graph G^*. Consequently, after we compute $\Phi(Q)$ it is possible to find a model for $\Phi(Q)$ in constant time. □

Finally, in Lemma 10.25 below we use Lemma 10.24 to replace any well-structured modulator by a small but "equivalent" modulator.

Lemma 10.25. *Let q be a non-negative integer constant and $\mathcal{H}$ be a graph class. Then given a graph G and a (k, c)-well-structured modulator $\vec{X} = \{X_1, \dots X_k\}$ of G into $\mathcal{H}$, there exists a function f such that one can in time $f(c) \cdot |V(G)|^{\mathcal{O}(1)}$ compute a graph G' with a (k, c)-well-structured modulator $\vec{X}' = \{X'_1, \dots X'_k\}$ into $\mathcal{H}$ such that $(G, \vec{X})$ and $(G', \vec{X}')$ are q-similar and for each $i \in [k]$ it holds that $|X'_i|$ is bounded by a constant.*

Proof. For $i \in [k]$, let $S_i \subseteq X_i$ be the frontier of split-module X_i (that is $S_i = \lambda(X_i)$), let $G_i = G[X_i]$ and let $G_0 = G \setminus G[\vec{X}]$. We compute a graph G'_i of constant size and a set $S'_i \subseteq V(G'_i)$ having the same MSO q-type as (G_i, S_i). By Lemma 10.24, this can be done in time $f(c) \cdot |V(G)|^{\mathcal{O}(1)}$ for some function f. Now let G' be the graph obtained by the following procedure:

1. We construct a disjoint union of G_0 and G'_i for each $i \in [k]$;

2. If $k \geq 2$ then for each $1 \leq i < j \leq k$ such that S_i and S_k are adjacent in G, we add edges between every $v \in S'_i$ and $w \in S'_j$.

3. for every $v \in V(G_0)$ and $i \in [k]$ such that S_i and $\{v\}$ are adjacent, we add edges between v and every $w \in S'_i$.

It is easy to verify that $(G, \vec{X})$ and $(G', \vec{X}')$, where $\vec{X}' = \{V(G'_1), \ldots, V(G'_k)\}$, are q-similar. □

10.4 Summary

In this chapter, we defined a novel family of parameters called well-structured modulators. Proposition 10.2 indicates that this family generalizes both rank-width and size of a modulator to a fixed graph class. Furthermore, we developed some machinery that will be important in the following tho chapters. Namely, the equivalence relation $\sim_k^G$ introduced in Section 10.2 plays a crucial role in algorithms for finding a well-structure modulator to various graph classes. On the other hand the notion of similarity and Lemma 10.25 allows us to replace the (k, c)-well-structured modulator by a modulator of size $\mathcal{O}(k)$, which allows us to obtain for many MSO definable problems FPT algorithms parameterized by $\mathrm{wsn}^{\mathcal{H}}(G)$ and even kernels parameterized by $\mathrm{wsn}_c^{\mathcal{H}}(G)$.

The results in this chapter appeared in a journal paper in Algorithmica [75] as well as in a conference paper in the proceedings of Algorithms and Data Structures 14th International Symposium (WADS 2015) [74].

CHAPTER 11

Fixed-Parameter Algorithms using Well-Structured Modulators

Here we further study the well-structured modulators introduced in the previous chapter. The main goal is to push the boundaries of tractability for many problems to also cover instances that have unbounded treewidth and even rank-width. In particular, we establish that under two necessary conditions, namely

1. we can find a good well-structured modulator for a class $\mathcal{H}$, and
2. there is an FPT algorithm for the given MSO_1 definable problem parameterized by the size of the modulator to $\mathcal{H}$,

the MSO_1 definable problem is FPT parameterized by $\text{wsn}^{\mathcal{H}}$.

Results

1. We develop a fixed-parameter algorithm for computing $\text{wsn}^{\mathcal{H}}$.

As with most structural parameters, virtually all algorithmic applications of the well-structure number rely on having access to an appropriate decomposition. We provide a fixed-parameter algorithm for computing the $\text{wsn}^{\mathcal{H}}$ along with the corresponding decomposition for any graph class $\mathcal{H}$ which can be characterized by a finite set of forbidden induced subgraphs (*obstructions*). Furthermore, we provide additional FPT algorithms for computing the $\text{wsn}^{\mathcal{H}}$ along with the corresponding decomposition if $\mathcal{H}$ is the class of all forests or the class of all chordal graphs. This is achieved by building on the structural results for split-modules from the previous chapter.

2. We design fixed-parameter algorithms for Vertex Cover and Clique parameterized by $\text{wsn}^{\mathcal{H}}$.

Specifically,we show that for any graph class $\mathcal{H}$ characterizable by a finite obstruction set and admitting a polynomial-time algorithm for Vertex Cover or Clique, there is a fixed-parameter algorithm solving Vertex Cover or Clique (respectively) when parameterized by $\mathrm{wsn}^{\mathcal{H}}$. We also give an overview of possible choices of $\mathcal{H}$ for Vertex Cover and Clique.

3. We develop a *meta-theorem* to obtain fixed-parameter algorithms for problems definable in MSO_1 parameterized by $\mathrm{wsn}^{\mathcal{H}}$.

The meta-theorem requires that the problem is FPT when parameterized by the cardinality of a modulator to $\mathcal{H}$. We prove that this condition is not only necessary but also tight, in the sense that the weaker condition of polynomial-time tractability on $\mathcal{H}$ used for Vertex Cover and Clique is not sufficient for FPT-time MSO model checking.

4. We show that, in general, solving MSO-Opt problems is not FPT when parameterized by $\mathrm{wsn}^{\mathcal{H}}$.

We give a proof that these problems remain NP-hard even on graphs of fixed $\mathrm{wsn}^{\mathcal{H}}$ under the same conditions as those used for MSO model checking. This is somewhat surprising, since the fixed-parameter tractability of MSO optimization problems usually follows from the methods used for MSO model checking. On the other hand, there are strictly more classes of bounded width for our parameter than for rank-width and hence one cannot expect that every problem which is FPT parameterized by rank-width would remain FPT when parameterized by the well-structure number.

Organization of the Chapter

We begin the Chapter in Section 11.1 by designing FPT algorithm for finding k-well-structured modulators. Afterwards in Section 11.2, we give two examples how we can obtain FPT algorithms parameterized by $\mathrm{wsn}^{\mathcal{H}}$ for specific problems, where an FPT algorithm parameterized by the size of an minimum modulator to $\mathcal{H}$ is known. Then, we switch our focus on problems definable by constant size MSO_1 formula and in Section 11.3 we develop an FPT algorithm that can be used with any MSO_1 definable problem that is in FPT when parameterized by the cardinality of a modulator to $\mathcal{H}$. Finally, in Section 11.4 we show that this result does not extend towards MSO optimization problems.

11.1 Finding Well-Structured Modulators

The objective of the first part of this section is to prove the following theorem.

Theorem 11.1. *Let $\mathcal{H}$ be a graph class characterized by a finite obstruction set $\mathcal{F}$. There exists a fixed-parameter algorithm parameterized by k which for every input graph G either finds a k-well-structured modulator to $\mathcal{H}$, or detects that no such k-well-structured modulator exists.*

We first present the algorithm and then show its running time and correctness.

Algorithm 11.1: FindWSM$_{\mathcal{F}}$

```
Input  : $k \in \mathbb{N}_0$, $n$-vertex graph $G$, equivalence $\sim$ over a superset of $V(G)$
Output : A $k$-cardinality set $\vec{X}$ of subsets of $V(G)$, or False

1  if $G$ does not contain any $D \in \mathcal{F}$ as an induced subgraph then
2      return $\emptyset$
3  else
4      $D' :=$ an induced subgraph of $G$ isomorphic to some $D \in \mathcal{F}$;
5  end
6  if $k = 0$ then return False;
7  foreach $[a]_\sim$ of $G$ which intersects with $V(D')$ do
8      $\vec{X} = \text{FindWSM}_{\mathcal{F}}(k-1, G - [a]_\sim, \sim)$;
9      if $\vec{X} \neq$ False then
10         return $\vec{X} \cup \{[a]_\sim\}$
11     end
12 end
13 return False
```

We will use $\sim_k$ (recall Definition 10.10) as the input for *FindWSM*$_{\mathcal{F}}$, however considering general equivalence relations as inputs is useful for proving correctness. Recall that the equivalence relation $\sim_k$ (or, more precisely, the set of its equivalence classes) can be computed in time $n^2 \cdot f(k) \cdot n^3$ for some function f thanks to Proposition 10.15, and this only needs to be done once before starting the algorithm. The following two lemmas show that Algorithm 11.1 is correct and runs in FPT time. For fixed $\mathcal{F}$, let $c_{\mathcal{F}}$ denote the maximum number of vertices of a graph in $\mathcal{F}$.

Lemma 11.2. FindWSM$_{\mathcal{F}}$ *runs in time* $\mathcal{O}(c_{\mathcal{F}}^k \cdot n^{c_{\mathcal{F}}})$.

Proof. The time required to perform the steps on lines **2-6** is $\mathcal{O}(n^{c_{\mathcal{F}}})$ since $\mathcal{F}$ is finite. Similarly, it holds that $|V(D')|$ and hence also the number of times the procedure on lines **8-13** is called are bounded by $c_{\mathcal{F}}$.

For the rest of the proof, we proceed by induction on k. First, if $k = 0$, then the algorithm is polynomial by the above. So assume that $k \geq 1$ and the algorithm for $k-1$ runs in time at most $c_{\mathcal{F}}^{k-1} \cdot n^{c_{\mathcal{F}}}$. Then the algorithm for k will run in polynomial time up to lines **8-13**, where it will make at most $c_{\mathcal{F}}$ calls to the algorithm for $k-1$, which implies that the running time for k is bounded by $\mathcal{O}(c_{\mathcal{F}}^k \cdot n^{c_{\mathcal{F}}})$. □

Lemma 11.3. *Let $k \geq 0$, G be a graph and $\sim$ be an equivalence relation over a superset of V. Then* $\text{FindWSM}_{\mathcal{F}}(k, G, \sim)$ *outputs a set $\vec{X}$ of at most k equivalence classes of $\sim$ such that $G - \vec{X}$ is $\mathcal{F}$-free.*

Proof. If G does not contain any D as an induced subgraph, then we correctly return the empty set. So, assume there exists an induced subgraph D' of G isomorphic to D. We prove the lemma by induction on k.

Clearly, if $k = 0$ but there exists some obstruction, then the algorithm outputs *False* and this is correct; if $k = 0$ and no obstruction exists, then the algorithm correctly outputs $\emptyset$. Let $k \geq 1$ and assume that the algorithm is correct for $k - 1$. If G does not contain any such $\vec{X}$, then for any equivalence class $[a]_\sim$, $\text{FindWSM}_{\mathcal{F}}(k - 1, G - [a]_\sim, \sim)$ will correctly output *False*.

On the other hand, assume G does contain some $\vec{X}$ with the desired properties. In particular, this implies that $\vec{X}$ must intersect $V(D')$. Let X_i be an arbitrary equivalence class of $\vec{X}$ which intersects $V(D')$. Then $\vec{X}' \setminus \{X_i\}$ is a set of at most $k - 1$ equivalence classes of $\sim$ in $G - X_i$, and hence $\text{FindWSM}_{\mathcal{F}}(k - 1, G - X_i', \sim)$ will output some solution $\vec{X}''$ for $G - X_i'$ by our inductive assumption. Since any obstruction in G intersecting X_i' is removed by X_i' and $G - X_i'$ is made $\mathcal{F}$-free by $\vec{X}''$, we observe that $\vec{X}'' \cup X_i'$ intersects every obstruction in G and hence the proof is complete. □

From Lemma 11.3 and Corollary 10.13 we obtain the following.

Corollary 11.4. *Let $k \in \mathbb{N}$, G be a graph and $\sim_k$ be the equivalence relation computed by Proposition 10.15. Then* $\text{FindWSM}_{\mathcal{F}}(k, G, \sim_k)$ *either outputs a k-well-structured modulator to $\mathcal{H}$ or correctly detects that no such modulator exists.*

Proof of Theorem 11.1. The theorem follows by using Proposition 10.15 and then Algorithm 11.1 in conjunction with Lemma 11.2 and Corollary 11.4. □

11.1.1 Results for Other Graph Classes

Next, we turn our attention to computing k-well-structured modulators to examples of graph classes which are not characterized by a finite obstruction set (i.e., by a finite set of forbidden induced subgraphs). In the following lemmas, n denotes the size of the vertex set of the input graph.

Lemma 11.5. *It is possible to compute a k-well-structured modulator to the class of forests in time $f(k) \cdot n^5$ for some computable function f.*

Proof. We begin by describing our algorithm $\mathcal{A}$, and then proceed to argue correctness and runtime bounds. $\mathcal{A}$ begins by checking whether the rank-width of the input graph G is at least $k + 2$; if not, then a k-well-structured modulator can be computed using Courcelle's Theorem in time at most $f(k) \cdot n^3$ for some computable function f. $\mathcal{A}$ then proceeds in four steps.

- First, it uses Proposition 10.15 to partition $V(G)$ into equivalence classes of $\sim_k$ in time at most $f(k) \cdot n^5$ for some computable function f, and sets $j := k$; $S := \emptyset$; $\sim := \sim_k$.
- Second, for each tuple (X, Y, Z) of equivalence classes of $\sim$, $\mathcal{A}$ checks whether $G[X \cup Y \cup Z]$ is acyclic. If this is not the case, then $\mathcal{A}$ chooses (by branching) one class out of $\{X, Y, Z\}$ to delete from $\sim$, saves the deleted equivalence class in S, and restarts the second step with $j := j - 1$. If $j = -1$, then the algorithm terminates the given branch.
- Third, $\mathcal{A}$ constructs an auxiliary graph $G' = (V', E')$ by setting V' to be the set of equivalence classes of $\sim$ and E' to contain an edge between $A, B \in V'$ iff there exist vertices $a \in A, b \in B$ such that $ab \in E(G)$.
- Finally, $\mathcal{A}$ tries to find a feedback vertex set in G' of size at most j in time $\mathcal{O}(3.83^j \cdot j|V'|^2)$ [39]. If no such feedback vertex set exists, then $\mathcal{A}$ terminates the given branch; otherwise it adds the feedback vertex set to $\vec{S}$ and outputs $\vec{S}$.

It is easy to verify that the steps two to four can be implemented in time $\mathcal{O}(n^4 + 3.83^k \cdot kn^2)$. Hence, the running time of $\mathcal{A}$ is upper-bounded by $\mathcal{O}(f(k) \cdot n^5)$ for some computable function f. As for correctness, let us assume for a contradiction that $\mathcal{A}$ outputs a set $\vec{S}$ and the graph H obtained from G after deleting all vertices in elements of $\vec{S}$ contains a cycle C. Clearly, neither C nor any other cycle in H intersects less than 4 equivalence classes of $\sim_k$, since otherwise such a cycle would have been detected and removed in step 2 of $\mathcal{A}$.

Moreover, assume $|C \cap X| > 1$ for some equivalence class X of $\sim_k$. Since C spans at least 4 equivalence classes, H must contain at least two neighbors of X in $C \setminus X$ which are adjacent to at least two vertices in X (indeed, recall that X is a split-module and hence all vertices of X with a neighbor outside X have the same neighborhood outside X); let us denote these vertices y, z, x_1, x_2, respectively. Since x_1, x_2 are in the frontier of X, the vertices y, x_1, z, x_2 must form a cycle in H which spans at most 3 equivalence classes, contradicting our previous conclusion that no such cycles are present in H. Hence we may conclude that $|C \cap X| \leq 1$ for every equivalence class X.

The only case we are left with now is that C intersects each equivalence class at most once. But then it must be the case that C also forms a cycle in G', which would have necessarily been removed in step 4 of $\mathcal{A}$, a contradiction. So H must indeed be acyclic.

For the other direction, assume that G contains a minimal k-well-structured modulator $\vec{X} = \{X_1, \ldots, X_j\}$ to the class of forests. Then consider the branch of step 2 of $\mathcal{A}$ which hits a maximal number of elements of $\vec{X}$, and let us denote the elements removed by $\mathcal{A}$ in this way $\vec{Y}$. By the same argument as above, each cycle remaining in G after deleting $\vec{Y}$ intersects each equivalence class at most once and hence corresponds to a cycle in the graph G' constructed by $\mathcal{A}$. In particular, the equivalence classes in $\vec{X} \setminus \vec{Y}$ form a feedback vertex set of size $\ell = |\vec{X} \setminus \vec{Y}|$ in G'. By the correctness of the feedback vertex set algorithm used in step 4, at least one branch of $\mathcal{A}$ is guaranteed to output a solution $\vec{S} \supset \vec{Y}$ of size at most j. □

For the next result, recall that a cycle is chordless if it is also an induced cycle of length at least 4, and a graph is chordal if it contains no chordless cycles.

Lemma 11.6. *It is possible to compute a k-well-structured modulator to the class of chordal graphs in time $f(k) \cdot n^{\mathcal{O}(1)}$.*

Proof. We once again first describe our algorithm $\mathcal{A}$. $\mathcal{A}$ begins by checking whether the rank-width of the input graph G is at least $k+2$; if not, then a k-well-structured modulator can be computed using Courcelle's Theorem in time at most $f(k) \cdot n^3$ for some computable function f. $\mathcal{A}$ then proceeds in four steps.

- First, it uses Proposition 10.15 to partition $V(G)$ into equivalence classes of $\sim_k$ in time at most $f(k) \cdot n^5$, and sets $j := k$; $S := \emptyset$; $\sim := \sim_k$.
- Second, for each tuple (X, Y, Z) of equivalence classes of $\sim$, $\mathcal{A}$ checks whether $G[X \cup Y \cup Z]$ is chordal in linear time [181]. If this is not the case, then $\mathcal{A}$ chooses (by branching) one class out of $\{X, Y, Z\}$ to delete from $\sim$, saves the deleted equivalence class in S, and restarts the second step with $j := j - 1$. If $j = -1$, then the algorithm terminates the given branch.
- Third, $\mathcal{A}$ constructs an auxiliary graph $G' = (V', E')$ by setting V' to be the set of equivalence classes of $\sim$ and E' to contain an edge between $A, B \in V'$ iff there exist vertices $a \in A, b \in B$ such that $ab \in E(G)$.
- Finally, $\mathcal{A}$ tries to find a modulator to chordal graphs of size at most j in G' using the algorithm by Marx [40], which takes time at most $2^{\mathcal{O}(k \log k)} \cdot |V'|^{\mathcal{O}(1)}$. If no such modulator exists, then $\mathcal{A}$ terminates the given branch; otherwise it adds the modulator to $\vec{S}$ and outputs $\vec{S}$.

It is easy to verify that the running time of $\mathcal{A}$ is upper-bounded by $f(k) \cdot n^{\mathcal{O}(1)}$ for some computable function f. As for correctness, let us assume for a contradiction that $\mathcal{A}$ outputs a set $\vec{S}$ and the graph H obtained from G after deleting all vertices in elements of $\vec{S}$ contains a chordless cycle C; without loss of generality, let us assume C is such a chordless cycle of minimum length. Clearly, neither C nor any other chordless cycle in H intersects less than 4 equivalence classes of $\sim_k$, since otherwise such a cycle would have been detected and removed in step 2 of $\mathcal{A}$.

We now claim that C contains at most one vertex from each equivalence class of $\sim_k$. To see this, assume for a contradiction that C contains two vertices in some equivalence class Z. Since C must also intersect other equivalence classes, it follows that C must in fact contain at least two vertices, say x, y, in the frontier of Z which have distinct neighbors, say x', y', respectively, in $C \setminus Z$. First, observe that x, y cannot occur consecutively along

C, as that would violate the assumption that C is a minimum-length chordless cycle. Hence by the chordality of C we also see that xy cannot be an edge of G, and the same also applies for the non-edge of $x'y'$. But then x', x, y', y forms a chordless cycle which intersects at most 3 equivalence classes, contradicting our previous assumptions.

Let us now consider the set C' of equivalence classes which intersect C; recall that $|C'| > 3$. By the above claim, it follows that C' would also be a chordless cycle in $V(G')$, contradicting the correctness of the chordal vertex deletion algorithm [40]. Hence we conclude that H must in fact be chordal.

For the other direction, assume that G contains a minimal k-well-structured modulator $\vec{X} = \{X_1, \ldots, X_j\}$ to the class of chordal graphs. Then consider the branch of step 2 of $\mathcal{A}$ which hits a maximal number of elements of $\vec{X}$, and let us denote the elements removed by $\mathcal{A}$ in this way $\vec{Y}$. By the same argument as above, each chordless cycle remaining in G after deleting $\vec{Y}$ intersects each equivalence class at most once and hence corresponds to a chordless cycle in the graph G' constructed by $\mathcal{A}$. In particular, the equivalence classes in $\vec{X} \setminus \vec{Y}$ form a modulator to chordal graphs of size $\ell = |\vec{X} \setminus \vec{Y}|$ in G'. By the correctness of the chordal vertex deletion algorithm used in step 4, at least one branch of $\mathcal{A}$ is guaranteed to output a solution $\vec{S} \supset \vec{Y}$ of size at most j. $\square$

11.2 Examples of Algorithmic Applications

This section contains two examples of how the notion of k-well-structured modulators can be used to design fixed-parameter algorithms. Our examples deal with two classical NP-hard graph problems, specifically Vertex Cover and Clique. Establishing the following theorem is the main objective of this subsection.

Theorem 11.7. *Let $\mathcal{P} \in \{$Vertex Cover, Clique$\}$ and $\mathcal{H}$ be a graph class characterized by a finite obstruction set. Then $\mathcal{P}$ is* FPT *parameterized by* $\text{wsn}^{\mathcal{H}}$ *if and only if $\mathcal{P}$ is polynomial-time tractable on $\mathcal{H}$.*

Since $\text{wsn}^{\mathcal{H}}(G) = 0$ for any $\mathcal{F}$-free graph G, the “only if” direction is immediate; in other words, being polynomial-time tractable on $\mathcal{H}$ is clearly a necessary condition for being fixed-parameter tractable when parameterized by $\text{wsn}^{\mathcal{H}}(G)$. Below we prove that for the selected problems this condition is also sufficient.

Lemma 11.8. *If* Vertex Cover *is polynomial-time tractable on a graph class $\mathcal{H}$ characterized by a finite obstruction set, then* Vertex Cover *parameterized by* $\text{wsn}^{\mathcal{H}}$ *is* FPT.

Proof. Let $G = (V, E)$ be a graph and let $k = \text{wsn}^{\mathcal{H}}(G)$. We start by using Theorem 11.1 to compute a k-well-structured modulator $\vec{X} = \{X_1, \ldots, X_k\}$ in FPT time. For each $i \in [k]$, we let $A_i = \lambda(X_i)$ (i.e., A_i is the frontier of X_i) and we let $B_i = N(A_i)$.

Since for each $i \in [k]$ the graph $G[A_i \cup B_i]$ contains a complete bipartite graph, any vertex cover of G must be a superset of either A_i or B_i. We can branch over these options for each i in 2^k time; formally, we branch over all of the at most 2^k functions $f : [i] \to \{A, B\}$, and refer to these as *signatures*. Each vertex cover Y of G can be associated with at least one signature f, constructed in the following way: for each $i \in [k]$ such that $A_i \subseteq Y$, we set $f(i) = A$, and otherwise we set $f(i) = B$.

Our algorithm then proceeds as follows. For a graph G and a signature f, we construct a partial vertex cover $Z = \bigcup_{i\in[k]} f(i)$. We let $G' = G - Z$. Consider any connected component C of G'. If C intersects some X_i, then by the construction of Z it must hold that $C \subseteq X_i$. Hence it follows that C either has rank-width at most $k + 1$ (in the case $C \subseteq X_i$ for some i), or C is in $\mathcal{H}$ (if C does not intersect $\vec{X}$), or both. Then we find a minimum vertex cover for each connected component of G' independently, by either calling the known fixed-parameter algorithm [93] (if C has bounded rank-width) or the polynomial algorithm (if C is in $\mathcal{H}$) at most $|C|$ times. Let Z' be the union of the obtained minimum vertex covers over all the components of G', and let $Y_f = Z \cup Z'$. After branching over all possible functions f, we compare the obtained cardinalities of Y_f and choose any Y_f of minimum cardinality. Finally, we compare $|Y_f|$ and the value of m provided in the input.

We argue correctness in two steps. First, assume for a contradiction that G contains an edge e which is not covered by Y_f for some f. Then e cannot have both endpoints in G', since Y_f contains a (minimum) vertex cover for each connected component of G', but e cannot have an endpoint outside of G', since $Z \subseteq Y_f$. Hence each Y_f is a vertex cover of G.

Second, assume for a contradiction that there exists a vertex cover Y' of G which has a lower cardinality than the vertex cover found by the algorithm described above. Let f be the signature of Y'. Then it follows that $Z \subseteq Y'$, and since $Z \subseteq Y_f$, there would exist a component C of $G \setminus Z$ such that $|Y' \cap C| \leq |Y_f \cap C|$. However, this would contradict the minimality of $Z' \cap C = Y_f \cap C$. Hence we conclude that no such Y' can exist, and the algorithm is correct. □

We deal with the second problem below.

Lemma 11.9. *If* Clique *is polynomial-time tractable on a graph class $\mathcal{H}$ characterized by a finite obstruction set, then* Clique *parameterized by* $\text{wsn}^{\mathcal{H}}$ *is* FPT.

Proof. We begin in the same way as for Vertex Cover: let $G = (V, E)$ be a graph and let $k = \text{wsn}^{\mathcal{H}}(G)$. If $\text{rw}(G) \leq k + 2$, then we simply use known algorithms to solve the problem in FPT time [93]. Otherwise, we proceed by using Theorem 11.1 to compute a k-well-structured modulator $\vec{X} = \{X_1, \ldots, X_k\}$ in FPT time. For each $i \in [k]$, we let $A_i = \lambda(X_i)$ and we let $B_i = N(A_i)$.

Let $X_0 = G - \vec{X}$ and let $s \subseteq \{0\} \cup [k]$. Then any clique C in G can be uniquely associated with a *signature* s by letting $i \in s$ if and only if $X_i \cap C \neq \emptyset$. The algorithm proceeds by branching over all of the at most 2^{k+1} possible non-empty signatures s. If $|s| = 1$, then the algorithm simply computes a maximum-cardinality clique in X_s (by calling the respective FPT or polynomial algorithm at most a linear number of times) and stores it as Y_s.

If $|s| \geq 2$, then the algorithm makes two checks before proceeding. First, if $0 \in s$ then it constructs the set X'_0 of all vertices $x \in X_0$ such that x is adjacent to every A_i for $i \in s \setminus \{0\}$. If $X'_0 = \emptyset$ then the current choice of s is discarded and the algorithm proceeds to the next choice of s. Second, for every $a \neq b$ such that $a, b \in s \setminus \{0\}$ it checks that $X'_a = A_a$ and $X'_b = A_b$ are adjacent; again, if this is not the case, then we discard this choice of s and proceed to the next choice of s. Finally, if the current choice of s passed both tests then for each $i \in s$ we compute a maximum clique in each $G[X'_i]$ and save their union as Y_s. In the end, we choose a maximum-cardinality set Y_s and compare its cardinality to the value of m provided in the input.

We again argue correctness in two steps. First, assume for a contradiction that Y_s is not a clique, i.e., there exist distinct non-adjacent $a, b \in Y_s$. Since Y_s consists of a union of cliques within subsets of $X'_{i \in s}$, it follows that there would have to exist distinct $c, d \in s$ such that $a \in X'_c$ and $b \in X'_d$. This can however be ruled out for c or d equal to 0 by the construction of X'_0. Similarly, if c and d are both non-zero, then this is impossible by the second check which tests adjacency of every pair of X'_c and X'_d for every $c, d \in s$.

Second, assume for a contradiction that there exists a clique Y' in G which has a higher cardinality than the largest clique obtained by the above algorithm. Let s be the signature of Y'. If $|s| = 1$ then $|Y_s| \geq |Y'|$ by the correctness of the respective FPT or polynomial algorithm used for each X_s. If $|s| \geq 2$ then Y' may only intersect the sets X' constructed above for s. Moreover, if there exists $i \in [k] \cup \{0\}$ such that $|Y' \cap X'_i| > |Y_s \cap X'_i|$ then we again arrive at a contradiction with the correctness of the respective FPT or polynomial algorithms used for X'_i. Hence we conclude that no such Y' can exist, and the algorithm is correct. $\square$

Finally, let us review some concrete graph classes for use in Theorem 11.7. We use K_i, C_i and P_i to denote the i-vertex complete graph, cycle, and path, respectively. $2K_2$ denotes the disjoint union of two K_2 graphs. The fork, $K_{3,3}$-e, banner, twin-house and $T_{2,2,2}$ graphs are depicted in Figure 11.1.

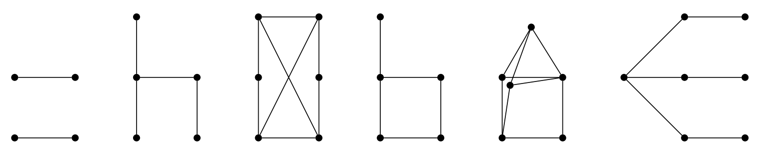


Figure 11.1: From left to right: $2K_2$, fork, $K_{3,3}$-e, banner, twin-house, and $T_{2,2,2}$.

Fact 11.10. Vertex Cover *is polynomial-time tractable on the following graph classes:*

1. *$(2K_2, C_4, C_5)$-free graphs (split graphs);*
2. *P_5-free graphs;*
3. *fork-free graphs;*
4. *$(banner, T_{2,2,2})$-free graphs and $(banner, K_{3,3}$-$e, twin\text{-}house)$-free graphs.*

Proof. For item 1, recall that split graphs are graphs whose vertex set can be partitioned into one clique and one independent set, and such a partitioning can be found in linear time. If each vertex in the clique is adjacent to at least one independent vertex, then the clique is a minimum vertex cover, otherwise the clique without a pendant-free vertex is a minimum vertex cover. Item 2 follows from [150]. Item 3 follows from [7]. Item 4 follows from [103] and [32]. □

Fact 11.11. Clique *is polynomial-time tractable on the following graph classes:*

1. *Any complementary graph class to the classes listed in Fact 11.10 (such as cofork-free graphs and split graphs);*
2. *Graphs of bounded degree.*

Proof.
1. It is well-known that each maximum clique corresponds to a maximum independent set (and vice-versa) in the complement graph.
2. The degree bounds the size of a maximum clique, again resulting in a simple folklore branching algorithm. The class of graphs of degree at most d is exactly the class of $\mathcal{F}$-free graphs for $\mathcal{F}$ containing all $(d+1)$-vertex supergraphs of the star having d leaves. □

11.3 MSO Model Checking with Well-Structured Modulators

Here we show how well-structured modulators can be used to solve the MSO model checking problem, as formalized in Theorem 11.12 below. Note that our meta-theorem captures not only the generality of MSO model checking problems, but also applies to a potentially unbounded number of choices of the graph class $\mathcal{H}$. Thus, the meta-theorem supports two dimensions of generality.

Theorem 11.12. *Let ϕ be a MSO_1 sentence, f, g be computable functions, and $\mathcal{H}$ be a graph class such that there exists an algorithm which finds a $g(\text{wsn}^{\mathcal{H}})$-well-structured modulator to $\mathcal{H}$ in time $f(k) \cdot |V|^{\mathcal{O}(1)}$. If* MSO-MC$_\phi$ *is* FPT *parameterized by $mod^{\mathcal{H}}(G)$, then* MSO-MC$_\phi$ *is also* FPT *parameterized by* $\text{wsn}^{\mathcal{H}}(G)$.

Proof. Let G be a graph, $k = \text{wsn}^{\mathcal{H}}(G)$ and q be the nesting depth of quantifiers in ϕ. By our assumption it is possible to find a $g(k)$-well-structured modulator to $\mathcal{H}$ in time $f(k) \cdot |V|^{\mathcal{O}(1)}$. We proceed by constructing $(G', \vec{X}')$ by Lemma 10.25. Since each $X'_i \in \vec{X}'$ has size bounded by a constant and $|\vec{X}'| \leq g(k)$, it follows that $\bigcup \vec{X}'$ is a modulator to the class of $\mathcal{F}$-free graphs of cardinality $\mathcal{O}(g(k))$. Hence MSO-MC$_\phi$ can be decided in FPT time on G'. Finally, since G and G' are q-similar, it follows from Lemma 10.23 that $G \models \phi$ if and only if $G' \models \phi$. □

Combining Theorem 11.12 with the results from Section 11.1, we obtain following corollaries.

Corollary 11.13. *Let ϕ be an MSO_1 sentence and $\mathcal{H}$ a graph class characterized by a finite obstruction set such that* MSO-MC$_\phi$ *is* FPT *parameterized by $\mathit{mod}^{\mathcal{H}}(G)$, the problem* MSO-MC$_\phi$ *is* FPT *parameterized by* $\text{wsn}^{\mathcal{H}}(G)$.

Corollary 11.14. *Let ϕ be an MSO_1 sentence and $\mathcal{H}$ the graph class of all forests such that* MSO-MC$_\phi$ *is* FPT *parameterized by $\mathit{mod}^{\mathcal{H}}(G)$, the problem* MSO-MC$_\phi$ *is* FPT *parameterized by* $\text{wsn}^{\mathcal{H}}(G)$.

Corollary 11.15. *Let ϕ be an MSO_1 sentence and $\mathcal{H}$ the graph class of all chordal graphs such that* MSO-MC$_\phi$ *is* FPT *parameterized by $\mathit{mod}^{\mathcal{H}}(G)$, the problem* MSO-MC$_\phi$ *is* FPT *parameterized by* $\text{wsn}^{\mathcal{H}}(G)$.

The condition that MSO-MC$_\phi$ is FPT parameterized by $\mathit{mod}^{\mathcal{H}}(G)$ is a necessary condition for the theorem to hold by Proposition 10.2. However, it is natural to ask whether it is possible to use a weaker necessary condition instead, specifically that MSO-MC$_\phi$ is polynomial-time tractable on $\mathcal{H}$ (as was done for specific problems in Section 11.2). Before proceeding towards a proof of Theorem 11.12, we make a digression and show that the weaker condition used in Theorem 11.7 is in fact not sufficient for the general case of MSO model checking.

Theorem 11.16. *There exists an MSO_1 sentence ϕ and a graph class $\mathcal{H}$ characterized by a finite obstruction set such that* MSO-MC$_\phi$ *is polynomial-time tractable on $\mathcal{H}$ but NP-hard on the class of graphs having* $\text{wsn}^{\mathcal{H}}(G) \leq 2$ *and even $\mathit{mod}^{\mathcal{H}}(G) \leq 2$.*

Proof. Consider the sentence ϕ which describes the existence of a proper 5-coloring of the vertices of G, and let $\mathcal{H}$ be the class of graphs of degree at most 4 (in other words, let $\mathcal{F}$ contain all 6-vertex supergraphs of the star having 5 leaves). There exists a trivial greedy algorithm to obtain a proper 5-coloring of any graph of degree at most 4, hence MSO-MC$_\phi$ is polynomial-time tractable on $\mathcal{H}$. Now consider the class of graphs obtained from $\mathcal{H}$ by adding, to any graph in $\mathcal{H}$, two adjacent vertices y, z which are both adjacent to every other vertex in the graph. By construction, any graph G' from this new class satisfies $\mathit{mod}^{\mathcal{H}}(G') \leq 2$ and hence also $\text{wsn}^{\mathcal{H}}(G') \leq 2$. However, G' admits a proper 5-coloring if and only if $G' - \{y, z\}$ admits a proper 3-coloring. Testing 3-colorability on graphs of degree at most 4 is known to be NP-hard [144]. □

We conclude the section by showcasing an example application of Theorem 11.12. c-Coloring asks whether the vertices of an input graph G can be colored by c colors so that each pair of neighbors have distinct colors. From the connection between c-Coloring, its generalization List c-Coloring and modulators [36, Theorem 3.3] and tractability results for List c-Coloring [107, Page 5], we obtain the following.

Corollary 11.17. *For each $c \in \mathbb{N}$, c-Coloring parameterized by* $\mathrm{wsn}^{P_5\text{-}free}$ *is* FPT.

11.4 Hardness of MSO Optimization

In the wake of Theorem 11.12 and the positive results for the two problems in Section 11.2, one would expect that it should be possible to strengthen Theorem 11.12 to also cover MSO-Opt problems [50, 93], which extend MSO model checking by allowing the minimization/maximization. Surprisingly this is not possible if we wish to retain the same necessary conditions, as will be shown in this section.

We say that $S \subseteq V(G)$ is a dominating set if every vertex in G either is in S or has a neighbor in S. We will need the following lemma before we proceed to the main result of this secton.

Lemma 11.18. *The problem of finding a p-cardinality dominating set in a graph G having a k-cardinality modulator $X \subseteq V(G)$ to the class of graphs of degree at most 3 is FPT when parameterized by $p + k$.*

Proof. Let $L = V(G) \setminus X$ and consider the following algorithm. We begin with $D = \emptyset$, and choose an arbitrary vertex $v \in L$ which is not yet dominated by D. We branch over the at most $k + 4$ vertices q in $\{v\} \cup N(v)$, and add q to D. If $|D| = p$ and there still exists an undominated vertex in G, we discard the current branch; hence this procedure produces a total of at most $(k + 4)^p$ branches.

Now consider a branch where $|D| < p$ but the only vertices left to dominate lie in X. For $a, b \in L$, we let $a \equiv b$ if and only if $N(a) \cap X = N(b) \cap X$. Notice that $\equiv$ has at most 2^k equivalence classes and that these may be computed in polynomial time. For each non-empty equivalence class of $\equiv$, we choose an arbitrary representative and construct the set P of all such chosen representatives. We then branch over all subsets Q of $P \cup X$ of cardinality at most $p - |D|$, and add Q into D. Since $|P \cup X| \leq 2^k + k$, this can be done in time bounded by $\mathcal{O}(2^{p \cdot k})$. Finally, we test whether this D is a dominating set, and output the minimum dominating set obtained in this manner.

It is easily observed from the description that the running time is FPT. For correctness, from the final check it follows that any set output by the algorithm will be a dominating set. It remains to show that if there exists a dominating set of cardinality p, then the algorithm will find such a set. So, assume there exists a p-cardinality dominating set D' in G. Consider the branch arising from the first branching rule obtained as follows. Let v_1 be the first undominated vertex in L chosen by the algorithm, and consider the branch where an arbitrary $q \in D' \cap N(v_1)$ is placed into D. Hence, after the first branching, there is a branch where $D \subseteq D'$. Similarly, there exists a branch where $D \subseteq D'$ for each v_i chosen in the i-th step of the first branching. If $D' = D$ after the first branching, then we are done; so, let $D'_1 = D' \setminus D$ be non-empty. Let D_1 be obtained from D'_1 by replacing each $w \in D'_1$ by the representative of $[w]_\equiv$ chosen to lie in P. Since D' dominates all vertices in L and D_1 dominates the same vertices in X as D'_1, it follows that $D^* = (D' \setminus D'_1) \cup D_1$ is also a dominating set of G. Furthermore, $|D^*| = |D'|$. However, since $D_1 \subseteq P$ and $|D_1| \leq p - |D|$, there must exist a branch in the second branching which sets $Q = D_1$. Hence there exists a branch in the algorithm which obtains and outputs the set $D^* = D \cup D_1$. □

Theorem 11.19. *There exists an* MSO_1 *formula* φ *and a graph class* $\mathcal{H}$ *characterized by a finite obstruction set such that* MSO-Opt$_\varphi^{\leq}$ *is* FPT *parameterized by mod*$^{\mathcal{H}}$ *but* paraNP*-hard parameterized by* wsn$^{\mathcal{H}}$.

Proof. To prove Theorem 11.19, we let $dom(S)$ express that S is a dominating set in G, and let $cyc(S)$ express that S intersects every C_4 (cycle of length 4). Then we set $\varphi(S) = dom(S) \vee cyc(S)$ and let $\mathcal{H}$ be the class of C_4-free graphs of degree at most 3 (obtained by letting the obstrucion set $\mathcal{F}$ contain C_4 and all 5-vertex supergraphs of $K_{1,4}$).

Claim 11.20. MSO-Opt$_\varphi^{\leq}$ *is FPT parameterized by the cardinality of a modulator to* $\mathcal{H}$.

To argue that the above claim holds, let $(G = (V, E), r)$ be the input of MSO-Opt$_\varphi^{\leq}$ and k be the cardinality of a modulator in G to $\mathcal{H}$. We begin by computing some modulator $X \subseteq V$ of cardinality k in G to $\mathcal{H}$; this can be done in FPT time by a simple branching algorithm on any of the obstructions from $\mathcal{F}$ located in G. Let $L = V \setminus X$. Next, we compare r and k, and if $r \geq k$ then we output YES. This is correct, since each C_4 in G must intersect X and hence setting $S = X$ satisfies $\varphi(S)$.

So, assume $r < k$. Then we check whether there exists a set A of cardinality at most r which intersects every C_4; this can be done in time $O^*(4^r)$ by a simple FPT branching algorithm. Next, we check whether there exists a dominating set B in G of cardinality at most r; this can also be done in FPT time by Lemma 11.18.

Finally, if A or B exists, then we output YES and otherwise we output NO. Hence the claim is indeed true.

Claim 11.21. MSO-Opt$_\varphi^{\leq}$ *is* paraNP*-hard parameterized by* wsn$^{\mathcal{H}}(G)$.

We proceed by arguing that this claim is also correct. It is known that the Dominating Set problem, which takes as input a graph G and an integer j and asks to find a dominating set of size at most j, is NP-hard on C_4-free graphs of degree at most 3 [145] (see also subsequent work [8, Theorem 8]). We use this fact as the basis of our reduction. Let (G, j) be a C_4-free instance of Dominating Set of degree at most 3. Then we construct G' from G by adding $(|G| + 2)$-many copies of C_4, a single vertex q adjacent to every vertex of every such C_4, and a single vertex q' adjacent to q and an arbitrary vertex of G. It is easy to check that $\mathrm{wsn}^{\mathcal{H}}(G') \leq 2$.

We claim that (G, j) is a YES-instance of Dominating Set if and only if $(G', j+1)$ is a yes-instance of MSO-Opt$^{\leq}_{\varphi}$. For the forward direction, assume there exists a dominating set D in G of cardinality j. Then the set $D \cup \{q\}$ is a dominating set in G', and hence satisfies φ.

On the other hand, assume there exists a set D' of cardinality at most $j+1$ which satisfies φ. If $j+1 \geq |G|+2$ then clearly (G, j) is a YES-instance of Dominating Set, so assume this is not the case. But then D' cannot intersect every C_4, and hence D' must be a dominating set of G' of cardinality at most $j+1$. But this is only possible if $q \in D'$. Furthermore, if $q' \in D'$, then replacing q' with the neighbor of q' in G is also a dominating set of G'. Hence we may assume, w.l.o.g., that $D' \cap V(G)$ is a dominating set of cardinality at most j in $V(G)$. Consequently, (G, j) is a YES-instance of Dominating Set and the claim holds. The theorem now follows from the two claims proved above. ☐

11.5 Summary and Open Questions

We showed here that well-structured modulators push the frontiers of fixed-parameter tractability beyond rank-width and modulator size for a wide range of problems. In particular, the well-structure number can be computed efficiently (Theorem 11.1) and used to design fixed-parameter algorithms for Vertex Cover and Clique (Theorem 11.7) as well as any problem which can be described by a sentence in MSO_1 logic (Theorem 11.12). We remark that while our results are of a theoretical nature, there is hope that some of the ideas behind the presented algorithms may be useful in practice once faster algorithms for computing rank-width become available.

For future work, it would be interesting to see whether the notion of split-modules introduced in this work can be naturally generalized. In particular, a split-module X can be seen as a subgraph such that $\boldsymbol{A}_G[X, G-X] = 1$[1], and in this sense split decompositions naturally correspond to rank-width 1. It is easy to define corresponding decompositions also for higher values of rank-width, however it is not at all clear how such decompositions could be computed. We believe this is an interesting question on its own; furthermore, obtaining such decompositions would allow an immediate extension of our framework to the arising more general notions of split-modules.

Notes

The results in this chapter appeared in a journal paper in Algorithmica [75] as well as in a conference paper in the proceedings of Algorithms and Data Structures 14th International Symposium (WADS 2015) [74].

[1]Recall that $\boldsymbol{A}_G[X, G-X]$ is the rank of the submatrix of the adjacency matrix of G over the field GF(2), where rows are restricted to the vertices in X and columns to the vertices in $V(G) \setminus X$.

CHAPTER 12

Meta-Kernelization using Well-Structured Modulators

In this chapter, we follow up on the recent line of research which studies meta-kernelization in terms of structural parameters. Gajarský et al. [92] developed a meta-kernelization framework parameterized by the size of a modulator to the class of graphs of bounded treedepth on sparse graphs. Ganian, Slivovsky and Szeider [97] independently developed a meta-kernelization framework using a different parameter based on rank-width and modular decompositions. Our results build upon both of the aforementioned papers by fully subsuming the meta-kernelization framework of Ganian, Slivovsky, and Szeider [97] and lifting the meta-kernelization framework of Gajarský et al. [92] to more general graph classes.

Results

1. We showcase applications of c-well-structured modulators on two special cases of graph classes generalizing meta-kernelization frameworks of Gajarsky et al. [92] and Ganian, Slivovsky, and Szeider [97].

We start by considering c-well-structured modulators to edgeless graphs, which generalize the *vertex cover number*. While it is known that there exist MSO_1-definable problems which do not admit a polynomial kernel parameterized by the vertex cover number on general graphs, on graphs of bounded expansion this is no longer the case (as follows for instance from the result by Gajarský et al. [92]). We prove that every MSO_1-definable problem admits a linear kernel parameterized by the well-structure number for edgeless graphs. In relation to the above, we also show in Theorems 12.1 and 12.4 that every MSO_1-definable problem admits a linear kernel parameterized by the c-well-structure number for the empty graph (without restriction on the expansion). We remark that these results represent a direct generalization of the meta-kernelization results by Ganian, Slivovsky, and Szeider [97]. The proof is based on a combination of the replacement techniques developed in Chapter 10 together with the annotation framework used by Ganian, Slivovsky, and Szeider [97].

2. We develop constant approximation algorithms for finding well-structured modulators to a plethora of graph classes.

Before we can proceed to wider applications of our parameter in kernelization, it is first necessary to deal with the subproblem of finding a suitable well-structured modulator in polynomial time. We resolve this question for well-structured modulators to a vast range of graph classes: We develop a 3-approximation algorithm for finding well-structured modulators to acyclic graphs. Furthermore, We give constant-factor approximation algorithms for well-structured modulators to graph classes characterized by a finite set of forbidden induced subgraphs. Armed with the approximation algorithms, we develop our most general result, which is the key for lifting kernelization results from modulators to well-structured modulators.

3. We prove that whenever a modulator to a graph class $\mathcal{H}$ can be used to obtain a polynomial kernel for some MSO_1-definable problem, this problem also admits a polynomial kernel when parameterized by the well-structure number for $\mathcal{H}$ as long as well-structured modulators to $\mathcal{H}$ can be approximated in polynomial time.

4. Finally, we show several applications of this theorem.

Since the class of graphs of treedepth bounded by some fixed integer can be characterized by a finite set of forbidden induced subgraphs, we use well-structured modulators to lift the results of Gajarsky et al. [92] from modulators to well-structured modulators for all MSO_1-definable decision problems (Theorem 12.27). Furthermore, by applying the *protrusion* machinery of Bodlaender et al. [26] and Kim et al. [138] we show that, in the case of bounded degree graphs, parameterization by a modulator to acyclic graphs (i.e., a feedback vertex set) admits the computation of a linear kernel for all MSO_1-definable decision problems. By our framework it then follows that such modulators can also be lifted to well-structured modulators (Theorem 12.30).

Organization of the Chapter

We start our investigations by a case study of well-structured modulators to two specific classes in Section 12.1, namely the class of edgeless graphs and class containing only the empty graph. Afterwards in Section 12.2, we develop approximation algorithms for finding well-structured modulators. Finally, Section 12.3 is devoted to showing our main kernelization results together with some applications.

Since $\text{wsn}^{\mathcal{H}}$ generalizes rank-width, we cannot hope to obtain kernels for k-well-structured modulators. Therefore, for the remainder of the chapter, we fix a constant c and focus our attention at (k, c)-well-structured modulators.

12.1 Case Studies

In this section we showcase how well-structured modulators can be used to obtain polynomial kernels for various problems.

12.1.1 Modulators to Empty Graphs

Let $\mathcal{Z}$ be the graph class containing only the empty graph. We remark that while $\text{mod}^{\mathcal{Z}}$ represents a very weak parameter as it is equal to the order of the graph, this is not the case for $\text{wsn}_c^{\mathcal{Z}}$. The goal of this subsection is to show how $\text{wsn}_c^{\mathcal{Z}}$ can be used for kernelization of MSO_1 Model Checking problems; this will later on be used to obtain a generalization of vertex cover as a parameter. This strictly generalizes the results of Ganian, Slivovsky, and Szeider [97], as their parameter, *rank-width$_c$ cover number of a graph G ($\text{rwc}_c(G)$)*, is the smallest number of *modules* the vertex set of G can be partitioned into such that each module induces a subgraph of rank-width at most c. Here module is a split-module such that every vertex in the split-module lies in its frontier. A wide range of problems, and in particular all MSO_1-definable problems, have been shown to admit linear kernels when parameterized by the rank-width$_c$ cover number [97].

Our proof strategy for this special case of well-structured modulators closely follows the replacement techniques used to obtain the kernelization results for the rank-width cover number [97], with the distinction that many of the tools and techniques had to be generalized to cover split-modules instead of modules.

Before we continue in this section, the reader is encouraged to recall the notion of *similarity* (Definition 10.22) together with Lemmas 10.25 and 10.23 which will be used in the proof of the following theorem.

Theorem 12.1. *Let $\mathcal{Z}$ be the class of empty graphs and $c \in \mathbb{N}$. For every MSO_1 sentence φ the problem* MSO-MC$_\varphi$ *admits a linear kernel parameterized by* $\text{wsn}_c^{\mathcal{Z}}$.

Proof. Let G be a graph, $k = \text{wsn}_c^{\mathcal{Z}}(G)$, and q be the nesting depth of quantifiers in ϕ. We use Proposition 10.15 to find the set $\vec{X}$ of equivalence classes of $\sim_c^G$ in polynomial time ($f(c) \cdot n^3$). Since equivalence classes of $\sim_c^G$ are unique and inclusion-maximal (see Corollary 10.13), the set $\vec{X}$ is a (k, c)-well-structured modulator to the empty graph.

We proceed by constructing $(G', \vec{X}')$ by Lemma 10.25. Since each $X_i' \in \vec{X}'$ has size bounded by a constant, $|\vec{X}'| \leq k$, and $\bigcup \vec{X}' = V(G')$, it follows that G' is an instance of MSO-MC$_\varphi$ of size $\mathcal{O}(k)$. Finally, since G and G' are q-similar, it follows from Lemma 10.23 that $G \models \phi$ if and only if $G' \models \phi$. □

Next, we combine the approaches used by Ganian, Slivovsky, and Szeider [97] and in Chapter 11 to handle MSO-$\text{Opt}_\varphi^\lozenge$ problems by using our more general parameter. Similarly as Ganian, Slivovsky, and Szeider [97], we use a more involved replacement procedure which explicitly keeps track of the original cardinalities of sets and results in an *annotated version* of MSO-$\text{Opt}_\varphi^\lozenge$. However, some parts of the framework (in particular the replacement procedure) had to be reworked using the techniques developed in Chapter 11, since we now use split-modules instead of simple modules. Given a graph $G = (V, E)$, an annotation $\mathcal{W}$ is a set of triples (X, Y, w) with $X \subseteq V, Y \subseteq V, w \in \mathbb{N}$. For every set $Z \subseteq V$ we define

$$\mathcal{W}(Z) = \textstyle\sum_{(X,Y,w)\in\mathcal{W}, X\subseteq Z, Y\cap Z=\emptyset} w.$$

The idea is that a triple (X, Y, w) assigns weight w to a vertex set X. Specifying the set Y allows us to control which subsets of Z the above sum is taken over. In the kernel, each set X will be a subset of a module M (with weight w corresponding to the optimum cardinality of a set in the matching module of the original graph). Setting $Y = M \setminus X$ ensures that the sum $\mathcal{W}(Z)$ contains at most one term for each module M. Each MSO formula $\varphi(X)$ and $\diamondsuit \in \{\leq, \geq\}$ gives rise to an annotated MSO-optimization problem.

*a*MSO-Opt$^{\diamondsuit}_{\varphi}$
Instance: A graph G with an annotation $\mathcal{W}$ and an integer $r \in \mathbb{N}$.
Question: Is there a set $Z \subseteq V(G)$ such that $G \models \varphi(Z)$ and $\mathcal{W}(Z) \diamondsuit r$?

Note that an instance of MSO-Opt$^{\diamondsuit}_{\varphi}$ can be represented as an instance of *a*MSO-Opt$^{\diamondsuit}_{\varphi}$ with the annotation $\mathcal{W} = \{ (\{v\}, \emptyset, 1) \mid v \in V(G) \}$. We call the pair $(G, \mathcal{W})$ an annotated graph. If the integer w is represented in binary, we can represent a triple (X, Y, w) in space $|X| + |Y| + \log_2(w)$. Consequently, we may assume that the size of the encoding of an annotated graph $(G, \mathcal{W})$ is polynomial in $|V(G)| + |\mathcal{W}| + \max_{(X,Y,w)\in\mathcal{W}} \log_2 w$.

Lemma 12.2. *Let $\varphi = \varphi(X)$ be a fixed MSO_1 formula. Then given an instance (G, r) of* MSO-Opt$^{\leq}_{\varphi}$ *and a (k, c)-well-structured modulator $\vec{X} = X_1, \ldots, X_k$ to $\mathcal{Z}$ of G, an annotated graph $(G', \mathcal{W})$ satisfying the following properties can be computed in polynomial time.*

1. *$(G, r) \in$ MSO-Opt$^{\leq}_{\varphi}$ if and only if $(G', \mathcal{W}, r) \in a$MSO-Opt$^{\leq}_{\varphi}$.*
2. *$|V(G')| \in \mathcal{O}(k)$.*
3. *The encoding size of $(G', \mathcal{W})$ is $\mathcal{O}(k \log(|V(G)|))$.*

Proof. Using Lemma 10.25 we compute a graph G' with a (k, c)-well-structured modulator $\{X'_1, \ldots, X'_k\}$ to $\mathcal{Z}$ such that $(G, \vec{X})$ and $(G', \vec{X}')$ are $(q+1)$-similar and $|X'_i|$ is bounded by a constant for each $i \in [k]$. To compute the annotation $\mathcal{W}$, we proceed as follows. For each $i \in [k]$, we go through all subsets $W' \subseteq X'_i$. By Lemma 10.18, we can compute a formula Φ such that for any graph H and $W \subseteq V(H)$ we have $\mathit{type}_q(G'[X'_i], W') = \mathit{type}_q(H, W)$ if and only if $H \models \Phi(W)$. Since $|X'_i|$ has constant size for every $i \in [k]$, this can be done within a constant time bound. Moreover, since $(G, \vec{X})$ and $(G', \vec{X}')$ are $(q+1)$-similar, there has to exist a $W \subseteq X_i$ such that $G[X_i] \models \Phi(W)$. Using Fact 3.9, we can compute a minimum-cardinality subset $W^* \subseteq X_i$ with this property in polynomial time. We then add the triple $(W', X'_i \setminus W', |W^*|)$ to $\mathcal{W}$. In total, the number of subsets processed is in $O(k)$. From this observation we get the desired bounds on the total runtime, $|V(G')|$, and the encoding size of $(G', \mathcal{W})$.

We claim that $(G', \mathcal{W}, r) \in a$MSO-Opt$^{\leq}_{\varphi}$ if and only if $(G, r) \in$ MSO-Opt$^{\leq}_{\varphi}$. Suppose there is a set $W \subseteq V(G)$ of vertices such that $G \models \varphi(W)$ and $|W| \leq r$. Since $X_1, \ldots, X_k$ is a partition of $V(G)$, we have $W = \bigcup_{i\in[k]} W_i$, where $W_i = W \cap X_i$. For each $i \in [k]$, let $W_i^* \subseteq X_i$ be a subset of minimum cardinality such that $\mathit{type}_q(G[X_i], W_i) = \mathit{type}_q(G[X_i], W_i^*)$. From the $(q+1)$-similarity of $(G, \vec{U})$ and $(G', \vec{U'})$, there is $W'_i \subseteq X'_i$ for each $i \in [k]$ such that $\mathit{type}_q(G'[X'_i], W'_i) = \mathit{type}_q(G[X_i], W_i^*)$. By construction, $\mathcal{W}$ contains a triple $(W'_i, X'_i \setminus W'_i, |W_i^*|)$. Observe that $(X, Y, w) \in \mathcal{W}$ and $(X, Y, w') \in \mathcal{W}$ implies $w = w'$. Let $W' = \bigcup_{i\in[k]} W'_i$. Then by $(q+1)$-similarity of $(G, \vec{X})$ and $(G', \vec{X'})$ and Lemma 10.23, we must have $\mathit{type}_q(G, W) = \mathit{type}_q(G', W')$. In particular, $G' \models \varphi(W')$. Furthermore,

$$\mathcal{W}(W') = \textstyle\sum_{(W'_i, X'_i \setminus W'_i, |W_i^*|)\in\mathcal{W}, X'_i \cap W' = W'_i} |W_i^*| \leq \sum_{i\in[k]} |W_i| = |W| \leq r.$$

For the converse, let $W' \subseteq V(G')$ such that $\mathcal{W}(W') \leq r$ and $G' \models \varphi(W')$, let W'_i denote $W' \cap X'_i$ for $i \in [k]$. By construction, there is a set $W_i \subseteq X_i$ for each $i \in [k]$ such that $\mathit{type}_q(G[X_i], W_i) = \mathit{type}_q(G'[X'_i], W'_i)$ and $\mathcal{W}(W') = \sum_{i\in[k]} |W_i|$. Let $W = \bigcup_{i\in[k]} W_i$. Then by congruence and Lemma 10.23 we get $\mathit{type}_q(G, W) = \mathit{type}_q(G', W')$ and thus $G \models \varphi(W)$. Moreover, $|W| = \mathcal{W}(W') \leq r$. □

The last thing we need to handle MSO-Opt$^{\leq}_{\varphi}$ problems is a win-win argument based on the following fact.

Fact 12.3 (Folklore)**.** *Given an* MSO_1 *sentence* φ *and a graph* G*, one can decide whether* $G \models \varphi$ *in time* $\mathcal{O}(2^{nl})$*, where* $n = |V(G)|$ *and* $l = |\varphi|$*.*

Theorem 12.4. *Let* $\mathcal{Z}$ *be the graph class containing only the empty graph and* $c \in \mathbb{N}$*. For every* MSO_1 *formula* φ *the problem* MSO-Opt$^{\leq}_{\varphi}$ *admits a linear bikernel parameterized by* $\mathrm{wsn}_c^{\mathcal{Z}}$*.*

Proof. We begin by proceeding similarly as in Theorem 12.1 to compute a (k, c)-well-structured modulator of the input graph G. Let $H_1, \ldots, H_j$ be the set of connected components of G. Then for each H_i of rank-width at least $c+2$, we can use Corollary 10.13 to find the set $\vec{X_i}$ of equivalence classes of $\sim_c^H$ in polynomial time. On the other hand, for each H_i of rank-width at most $c+1$, we set $\vec{X_i} = V(H_i)$. Clearly, the set $\vec{X} = \bigcup_{i\in[j]} \vec{X_i}$ is a (k, c)-well-structured modulator to the empty graph.

Let $(G', \mathcal{W})$ be the annotated graph computed from G and $\vec{X}$ according to Lemma 12.2. Let $n = |V(G)|$ and suppose $2^k \leq n$. Then we can solve $(G', \mathcal{W}, r)$ in time $n^{\mathcal{O}(1)}$. To do this, we go through all $2^{\mathcal{O}(k)}$ subsets W of $V(G')$ and test whether $\mathcal{W}(W) \leq r$. If that is the case, we check whether $G' \models \varphi(W)$. By Fact 12.3 this check can be carried out in time $c_1 2^{c_2 k} \leq c_1 n^{c2}$ for suitable constants c_1 and c_2 depending only on φ. Thus we can find a constant t such that the entire procedure runs in time n^t whenever n is large enough. If we find a solution $W \subseteq V(G')$ we return a trivial yes-instance; otherwise, a trivial no-instance (of aMSO-Opt$^{\leq}_{\varphi}$). Now suppose $n < 2^k$. Then $\log(n) < k$ and so the encoding size of $\mathcal{W}$ is polynomial in k. Thus $(G', \mathcal{W}, r)$ is a polynomial bikernel. □

12.1.2 Modulators to Edgeless Graphs

Our next order of business is to use well-structured modulators to generalize the use of vertex cover number as a structural parameter for kernelization. Let $\mathcal{E}$ be the graph class containing all edgeless graphs and $c \in \mathbb{N}$. It is easy to observe that $\mathrm{wsn}^{\mathcal{E}}_c$ is always upper-bounded by the vertex cover number (see also the following Proposition 12.5), and hence kernelization results obtained by using $\mathrm{wsn}^{\mathcal{E}}_c$ as a parameter represent a generalization of results obtained using the vertex cover number [87, 27]. We begin with a comparison to known structural parameters.

Proposition 12.5. *Let $\mathcal{E}$ be the graph class of edgeless graphs. Then:*

1. $\mathrm{rwc}_c(G) \geq \mathrm{wsn}^{\mathcal{E}}_c(G)$ *for any graph G. Furthermore, for every $i \in \mathbb{N}$ there exists a graph G_i such that* $\mathrm{rwc}_c(G_i) \geq 2i$ *and* $\mathrm{wsn}^{\mathcal{E}}_c \leq 2$.
2. $\mathrm{vcn}(G) \geq \mathrm{wsn}^{\mathcal{E}}_1(G)$ *for any graph G. Furthermore, for every $i \in \mathbb{N}$ there exists a graph G_i such that $vcn(G) \geq i$ and* $\mathrm{wsn}^{\mathcal{E}}_1 = 1$.

Proof. The first claim follows from the fact that rank-width cover is also a well-structured modulator to the empty graph. For the second claim, let G'_c be a graph of rank-width $c+1$, of bounded degree and of order at least i containing at least one vertex, say v, such that $G'_c - v$ has rank-width c. Next, we construct the graph G_c from $G'_c - v$ by exhaustively applying the following operation: for each module in the graph containing more than a single vertex, we create a new pendant and attach it to a single vertex in that module. Observe that this operation preserves the rank-width of the graph, and moreover the resulting graph only contains trivial modules (i.e., modules which contain a single vertex). Finally, let G^*_c be obtained from 2 disjoint copies of G_c, say G^1_c and G^2_c, and making the vertices which were adjacent to v in G^1_c adjacent to the vertices which were adjacent to v in G^2_c. Then $\mathrm{wsn}^{\mathcal{E}}_c(G^*_c) \leq 2$, since G^1_c and G^2_c are each a split-module of rank-width at most c. However, since G^*_c is a (vertex-)supergraph of G'_c, it follows that $\mathrm{rw}(G^*_c) \geq c+1$ and furthermore G^*_c only contains trivial modules. Hence $\mathrm{rwc}_c(G^*_c) \geq 2i$.

The third claim follows from the fact that any vertex cover of G is also a well-structured modulator to $\mathcal{E}$. For the fourth and final claim, consider a path P of length $2i+1$. Then $\mathrm{vcn}(P) \geq i$ but $\mathrm{wsn}^{\mathcal{E}}_1(P) = 1$. □

As we have established that already $\mathrm{wsn}^{\mathcal{E}}_1 \leq \mathrm{vcn}(G)$, it is important to mention that an additional structural restriction on the graph is necessary to allow the polynomial kernelization of MSO-Opt problems in general (as is made explicit in the following Fact 12.6).

Fact 12.6 ([29])**.** CLIQUE *parameterized by the vertex cover number does not admit a polynomial kernel, unless* $\mathsf{NP} \subseteq \text{coNP/poly}$.

However, it turns out that restricting the inputs to graphs of bounded expansion completely changes the situation: under this condition, it is not only the case that all MSO-MC and MSO-OPT problems admit a linear kernel when parameterized by the vertex cover number, but also when parameterized by the more general parameter $\text{wsn}_c^{\mathcal{E}}$. Before we can formally prove these claim, we need introduce the notion of sparseness and of bounded expansion.

Graph Sparsity

Definition 12.7 (Shallow minor [160], [92])**.** For any graphs H and G and any integer d, the graph H is said to be shallow minor of G at *depth d* if there exists a collection $\mathcal{P}$ of disjoint subsets $V_1, \ldots, V_p$ of $V(G)$ such that:

1. Each graph $G[V_i]$ has radius at most d: there exists in each set V_i a vertex x_i (a *center*) such that every vertex in V_i is at distance at most d from x_i in $G[V_i]$,
2. H is a subgraph of the graph resulting from G after contracting all edges in $\mathcal{P}$: There is an bijection $\psi : V(H) \to \mathcal{P}$, such that any two adjacent vertices u and v of H correspond to two sets $\psi(u)$ and $\psi(v)$ linked by at least one edge.

The set of all shallow minors of G at depth d is denoted by $G\triangledown d$. In particular, $G\triangledown 0$ is the set of all subgraphs of G. This notation is extended to graph classes $\mathcal{G}$ as well: $\mathcal{G}\triangledown d = \bigcup_{G\in\mathcal{G}} G\triangledown d$.

Definition 12.8 (Greatest reduced average density (grad) [160], [92])**.** Let $\mathcal{G}$ be a graph class. Then the greatest reduced average density of $\mathcal{G}$ with *rank d* is defined as

$$\nabla_d(\mathcal{G}) = \sup_{H\in\mathcal{G}\triangledown d} \frac{|E(H)|}{|V(H)|}.$$

Definition 12.9 (Bounded expansion [160])**.** A graph class $\mathcal{G}$ has bounded expansion if there exists a function $f : \mathbb{N} \to \mathbb{R}$ (called the *expansion function*) such that for all $d \in \mathbb{N}$, $\nabla_d(\mathcal{G}) \leq f(d)$.

We remark that on graphs of bounded expansion, our results could equivalently be stated in terms of treewidth (instead of rank-width) and MSO_2 logic (instead of MSO_1 logic). We briefly formalize this claim below.

We say that a class $\mathcal{H}$ of graphs is *uniformly k-sparse* if there exists k such that for every $G \in \mathcal{H}$ every finite subgraph of G has a number of edges bounded by k times the number of vertices. We note that *uniformly k-sparse* graphs are also studied under the name k-degenerate graphs [160].

Fact 12.10 ([47]). *For each integer k, one can effectively transform a given MSO_2 formula into an equivalent MSO_1 formula on finite, uniformly k-sparse, simple, directed or undirected graphs.*

Fact 12.11 ([47]). *A class of finite, uniformly k-sparse, simple, directed or undirected graphs has bounded tree-width if and only if it has bounded rank-width.*

Observation 12.12. *Every class of graphs of bounded expansion is uniformly k-sparse for some positive integer constant k.*

Proof. Let $\mathcal{H}$ be a class of graphs of bounded expansion and let f be the expansion function of $\mathcal{H}$. Then $f(0)$ or equivalently the greatest reduced average density of $\mathcal{H}$ with rank 0 is constant and is an exact upper bound on the ratio between the number of edges and vertices of any subgraph of a graph in $\mathcal{H}$. Therefore, $\mathcal{H}$ is uniformly $f(0)$-sparse. $\square$

Fact 12.13 ([92]). *Let $\mathcal{K}$ be a graph class with bounded expansion. Suppose that for $G \in \mathcal{K}$ and $S \subseteq V(G)$, $\mathcal{C}_1, \ldots, \mathcal{C}_s$ are sets of connected components of $G - S$ such that for all pairs $C, C' \in \bigcup_i \mathcal{C}_i$ it holds that $C, C' \in \mathcal{C}_j$ for some j if and only if $N_S(C) = N_S(C')$. Let $\delta \geq 0$ be a constant bound on the diameter of these components, i.e., for all $C \in \bigcup_i \mathcal{C}_i$, $diam(G[V(C)]) \leq \delta$. Then there can be only at most $\mathcal{O}(|S|)$ such sets $\mathcal{C}_i$.*

We note that δ in the above fact is a constant which is suppressed in $\mathcal{O}(|S|)$. This allows us to establish a key link between $\mathrm{wsn}_c^{\mathcal{E}}$ and $\mathrm{wsn}_c^{\mathcal{Z}}$ on graphs of bounded expansion.

Lemma 12.14. *Let $\mathcal{K}$ be a graph class with bounded expansion. Then there exists a constant d such that for every $G \in \mathcal{K}$ it holds that $\mathrm{wsn}_c^{\mathcal{Z}}(G) \leq d \cdot \mathrm{wsn}_c^{\mathcal{E}}(G)$.*

Proof. Let $k = \mathrm{wsn}_c^{\mathcal{E}}(G)$ and let $\vec{H}$ be a (k, c)-well-structured modulator to $\mathcal{E}$. Let S be a set of vertices containing exactly one vertex from the frontier of every split-module in $\vec{H}$. The graph $G' = G - (\vec{H} - S)$ is a graph with bounded expansion and S is its vertex cover. Clearly, the diameter of every connected component of $G' \setminus S$ is at most 1 (every connected component is a singleton). Therefore, by Fact 12.13 there exists a constant d' such that there are at most $d' \cdot |S| = d' \cdot \mathrm{wsn}_c^{\mathcal{E}}(G)$ sets of vertices $\mathcal{C}_1, \ldots, \mathcal{C}_s$ in $G' - S$ such that for all pairs $v, v' \in \bigcup_i \mathcal{C}_i$ it holds that $v, v' \in \mathcal{C}_j$ for some j if and only if $N_S(v) = N_S(v')$. Clearly each such $\mathcal{C}_i$ is a split-module in G'; furthermore, since each vertex in $V(G) \setminus V(G')$ has the same neighbors in $\mathcal{C}_i$ as some $s \in S$ (or alternatively may have no neighbors at all in $\mathcal{C}_i$), $\mathcal{C}_i$ is also a split-module in G. Moreover, each such $\mathcal{C}_i$ has rank-width at most 1. Hence $\mathrm{wsn}_c^{\mathcal{Z}}(G) \leq \mathrm{wsn}_c^{\mathcal{E}}(G) + d' \cdot \mathrm{wsn}_c^{\mathcal{E}}(G)$. $\square$

The above lemma allows us to shift our attention from modulators to $\mathcal{E}$ to a partition of the vertex set into split-modules of bounded rank-width, i.e., to well-structured modulators to $\mathcal{Z}$. In combination with Theorems 12.1 and 12.4, Lemma 12.14 immediately leads to the following.

Corollary 12.15. *Let $\mathcal{K}$ be a graph class of bounded expansion, $\mathcal{E}$ be the class of edgeless graphs and $c \in \mathbb{N}$. For every MSO_1 or MSO_2 sentence φ the problem* MSO-MC$_\varphi$ *admits a linear kernel parameterized by* $\mathrm{wsn}_c^{\mathcal{E}}$ *on $\mathcal{K}$. Furthermore, the problem* MSO-Opt$_\varphi^{\leq}$ *admits a linear bikernel parameterized by* $\mathrm{wsn}_c^{\mathcal{E}}$ *on $\mathcal{K}$.*

12.2 Finding (k, c)-Well-Structured Modulators

In this section we revisit the results of Section 11.1 and show that we can also find approximate well-structured modulators in polynomial time. Note that the algorithms in Section 11.1 could be easily modified to obtain approximation algorithms by simply replacing branching on which split-module of a constant number of split-modules intersect a solution by taking all of these split-modules to an approximate solution. However, we present here an alternative algorithms for approximating the c-well-structured number for the class of forests and then for any graph class which can be characterized by a finite set of forbidden induced subgraphs.

12.2.1 Finding (k, c)-Well-Structured Modulators to Forests

Our starting point is the following lemma, which shows that long cycles which hit a non-singleton frontier imply the existence of short cycles.

Lemma 12.16. *Let C be a cycle in G such that C intersects at least three distinct equivalence classes of $\sim_c$, one of which has a frontier of cardinality at least 2. Let Z be the set of equivalence classes of $\sim_c$ which intersect C. Then there exists a cycle C' such that the set Z' of equivalence classes it intersects is a subset of Z and $|Z'| \leq 3$.*

Proof. Let B be an equivalence class in Z such that $b_1, b_2 \in \lambda(B)$ are two distinct vertices. By assumption, C must contain two distinct vertices $a, c \notin B$ which are adjacent to $\lambda(B)$. Then a, b_1, c, b_2 forms a C_4 in $G[B \cup \{a, c\}]$. □

We will use the following observation to proceed when Lemma 12.16 cannot be applied.

Observation 12.17. *Assume that for each equivalence class B of $\sim_c$ it holds that $G[B]$ is acyclic, and that no cycle intersects B if $|\lambda(B)| \geq 2$. Then for every cycle C in G and every vertex $a \in C$, it holds that a is in the frontier of some equivalence class of $\sim_c$.*

Fact 12.18 below is the last ingredient needed for the algorithm.

Fact 12.18 ([20])**.** Feedback Vertex Set *can be 2-approximated in polynomial time.*

Theorem 12.19. *Let $c \in \mathbb{N}$ and $\mathcal{F}$ be the class of forests. There exists a polynomial algorithm which takes as input a graph G and computes a set $\vec{X}$ of split-modules such that $\vec{X}$ is a (k, c)-well-structured modulator to $\mathcal{F}$ and $k \leq 3 \cdot \mathrm{wsn}_c^{\mathcal{F}}$.*

Proof. First, consider the case that G is not connected. For a connected component H of G, let $\vec{X_H}$ be the restriction of $\vec{X}$ to split-modules contained in H. Then clearly each set $\vec{X}$ is a $(|\vec{X}|, c)$-well-structured modulator to $\mathcal{F}$ in G if and only if, for each connected component H of G, $\vec{X_H}$ is a $(|\vec{X_H}|, c)$-well-structured modulator to $\mathcal{F}$ in H. Therefore each connected component of G can be handled separately. Moreover, each connected component H of G of rank-width at most $c+1$ forms a single maximal split-module in any optimal (k, c)-well-structured modulator; if H is a tree, then we do not add it to $\vec{X}$, and otherwise we must add it to $\vec{X}$. Hence we can restrict our attention only to connected graphs G of rank-width at least $c+2$.

We now describe the algorithm for connected graphs G of rank-width at least $c+2$ and then argue its correctness. The algorithm proceeds in three steps.

I By deciding $a \sim_c b$ for each pair of vertices in G as per Fact 10.15, we compute the equivalence classes of $\sim_c$.

II For each set of up to three equivalence classes $\{A_1, A_2, A_3\}$ of $\sim_c$, we check if $G[A_1 \cup A_2 \cup A_3]$ is acyclic; if it's not, then we add A_1, A_2 and A_3 to $\vec{X}$ and set $G := G - (A_1 \cup A_2 \cup A_3)$.

III We use Fact 12.18 to 2-approximate a feedback vertex set S of G in polynomial time; let S' contain every equivalence class of $\sim_c$ which intersects S. We then set $\vec{X} := \vec{X} \cup S'$, and output $\vec{X}$.

For correctness, observe that Step III guarantees that $G - \vec{X}$ is acyclic. Hence we only need to argue that $|\vec{X}| \leq 3 \cdot \mathrm{wsn}_c^{\mathcal{F}}$. So, assume for a contradiction that there exists a (ℓ, c)-well-structured modulator $\vec{X'}$ to $\mathcal{F}$ such that $|\vec{X}| > 3 \cdot |\vec{X'}|$. Let Λ be the set of all equivalence classes of $\sim_c$ which were added to $\vec{X}$ in Step II of the algorithm. Since for each such $\{A_1, A_2, A_3\}$ the graph $G[A_1 \cup A_2 \cup A_3]$ is not acyclic, $\vec{X'}$ must always contain at least one split-module A' such that $A' \cap A_i \neq \emptyset$ for some $i \in [3]$. By Corollary 10.13 the relation $\sim_c$ is an equivalence and hence it follows that all the vertices in $A' \cup A_i$ are contained in a split-module of rank-width at most c. However, A_i is an inclusion-maximal split-module of rank-width at most c by Corollary 10.13 and therefore it follows that $A' \subseteq A_i$. Let Λ' contain all such split-modules A', i.e., all elements of $\vec{X'}$ which form a subset of a split-module added to $\vec{X}$ in Step II.

Let $\vec{X_3} = \vec{X} \setminus \Lambda$ and $\vec{X_3'} = \vec{X'} \setminus \Lambda'$. Since $|\Lambda| \leq 3 \cdot |\Lambda'|$ by the argument above, from our assumption it would follow that $|\vec{X_3}| > 3 \cdot |\vec{X_3'}|$. Let us consider the graphs $G_3 = G - \Lambda$ and $G_3' = G - \Lambda'$; observe that $G_3 \subseteq G_3'$. Furthermore, by Lemma 12.16 a cycle C in G_3 cannot intersect any equivalence class B of $\sim_c$ such that $\lambda(B) \geq 2$. Hence we can apply Observation 12.17, from which it follows that there is a one-to-one correspondence between any minimal feedback vertex set in G_3 and the equivalence classes of $\sim_c$ in G_3. Let z be the cardinality of a minimum feedback vertex set in G_3; by the correctness of the algorithm of Fact 12.18, we have $z \leq |\vec{X_3}| \leq 2z$. Since G_3' is a supergraph of G_3, it follows that $|\vec{X_3'}| \geq z$, and hence from our assumption we would obtain $2z \geq |\vec{X}_3| > 3 \cdot |\vec{X}_3'| \geq 3z$. We have thus reached a contradiction, and conclude that there exists no (ℓ, c)-well-structured modulator to $\mathcal{F}$ such that $|\vec{X}| > 3 \cdot \ell$. $\square$

12.2.2 Finding (k, c)-Well-Structured Modulators via Obstructions

Here we will show how to efficiently find a sufficiently small (k, c)-well-structured modulator to any graph class which can be characterized by a finite set of forbidden induced subgraphs. Let us fix a graph class $\mathcal{H}$ characterized by a set $\mathcal{R}$ of forbidden induced subgraphs, and let r be the maximum order of a graph in $\mathcal{R}$. Our first step is to reduce our problem to the classical Hitting Set problem.

Given a graph G, we construct an instance W_G of r-Hitting Set as follows. For each connected component H of G of rank-width at most $c + 1$, we add an element s_H representing H to the ground set S of W_G. On the other hand, for each connected component H of G of rank-width at least $c+2$, we add an element s_A for each equivalence class $A \subseteq V(G)$ of $\sim_c$ into S. Observe that S represents a partition of $V(G)$. Finally, for each induced subgraph $R \subseteq G$ isomorphic to an element of $\mathcal{R}$, we add the set $C_R = \{\, s_D \in S \mid D \cap R \neq \emptyset \,\}$ of elements which intersect R into $\mathcal{C}$. This completes the construction of W_G; we let $\mathrm{hit}(W_G)$ denote the cardinality of a solution of W_G.

Lemma 12.20. *For any graph G, the instance W_G is unique and can be constructed in polynomial time. Every hitting set Y in W_G is a $(|Y|, c)$-well-structured modulator to $\mathcal{H}$ in G. Moreover, $\mathrm{wsn}_c^{\mathcal{H}} = \mathrm{hit}(W_G)$.*

Proof. The uniqueness of W_G, as well as the fact that W_G can be constructed in polynomial time, follow from Corollary 10.13 together with the observation that all subgraphs $R \subseteq G$ isomorphic to an element of $\mathcal{R}$ can be enumerated in polynomial time.

For the second claim, consider a hitting set Y and let $\vec{Y}$ be the set equivalence classes of $\sim_c$ corresponding to the elements in Y. The graph $G - \vec{Y}$ cannot contain any obstruction for $\mathcal{H}$, and hence $G - \vec{Y} \in \mathcal{H}$.

For the third claim, assume G contains a $(\mathrm{hit}(W_G) - 1, c)$-well-structured modulator $\vec{X}$ to $\mathcal{H}$. By Corollary 10.13, each element A of $\vec{X}$ forms either a subset of an equivalence class A' of $\sim_c$ for some connected component of G, or a subset of a connected component of rank-width at most $c + 1$; in particular, this means that each such element A has a unique superset A' such that the element $s_{A'}$ is in the ground set of W_G. Let $\vec{X}'$ be obtained by replacing each element of A by its respective superset A'. Then $G - \vec{X}'$ is a subgraph of $G - \vec{X}$, and hence by our assumption $G - \vec{X}'$ cannot contain any subgraph isomorphic to an element of $\mathcal{R}$. However, this would imply that the set of elements of the ground set that correspond precisely to the equivalence classes in $\vec{X}'$ is a hitting set in W_G of cardinality $\mathrm{hit}(W_G) - 1$, which is a contradiction. $\square$

The final ingredient we need for our approximation algorithm is the following folklore result.

Fact 12.21 (Folklore, [139]). *There exists a polynomial-time algorithm which takes as input an instance W of r-*Hitting Set *and outputs a hitting set Y of cardinality at most $r \cdot \mathrm{hit}(W)$.*

We can now proceed to the main result of this subsection.

Theorem 12.22. *Let $c \in \mathbb{N}$ and $\mathcal{H}$ be a class of graphs characterized by a finite set of forbidden induced subgraphs of order at most r. There exists a polynomial algorithm which takes as input a graph G and computes a (k, c)-well-structured modulator to $\mathcal{H}$ such that $k \leq r \cdot \mathrm{wsn}_c^{\mathcal{H}}$.*

Proof. We proceed in two steps: first, we compute the r-Hitting Set instance W_G, and then we use Fact 12.21 to compute an r-approximate solution Y of W_G in polynomial time. We then set $\vec{X} := Y$ and output. Correctness follows from Lemma 12.20. In particular, the hitting set Y computed by Fact 12.21 has cardinality at most $r \cdot \mathrm{hit}(W_G)$ and hence $|\vec{X}| \leq r \cdot \mathrm{wsn}_c^{\mathcal{H}}(G)$ by Lemma 12.20. □

12.3 Applications of (k, c)-Well-Structured Modulators

We now proceed by outlining the general applications of our results. Our algorithmic framework is captured by the following Theorem 12.23.

Theorem 12.23. *Let p, q be polynomial functions, ϕ an MSO_1 sentence and $\mathcal{H}$ a graph class such that*

1. MSO-MC$_\phi$ *admits a (bi)kernel of size $p(\mathrm{mod}^{\mathcal{H}}(G))$, and*
2. *there exists a polynomial algorithm which finds a $(q(\mathrm{wsn}_c^{\mathcal{H}}), c)$-well-structured modulator to $\mathcal{H}$.*

Then MSO-MC$_\phi$ *admits a (bi)kernel of size $p(q(\mathrm{wsn}_c^{\mathcal{H}}(G)))$.*

Proof. Let G be a graph, $k = \mathrm{wsn}_c^{\mathcal{H}}(G)$ and s be the nesting depth of quantifiers in ϕ. We begin by computing a $(q(\mathrm{wsn}_c^{\mathcal{H}}), c)$-well-structured modulator to $\mathcal{H}$, denoted $\vec{X}$, in polynomial time by using Condition 2. We then proceed by constructing $(G', \vec{X}')$ by Lemma 10.25. Since each $X'_i \in \vec{X}'$ has size bounded by a constant and $|\vec{X}'| \leq k$, it follows that $\bigcup \vec{X}'$ is a modulator to $\mathcal{H}$ graphs of cardinality $\mathcal{O}(q(k))$. Then, using Condition 1, MSO-MC$_\phi$ admits a kernel of size $p(q(k))$ on G'. Finally, since G and G' are q-similar, it follows from Lemma 10.23 that $G \models \phi$ if and only if $G' \models \phi$. □

Similarly, as in the previous chapter, Theorem 11.16 implies that the condition that MSO-MC$_\phi$ admits a polynomial (bi)kernel parameterized by $\mathrm{mod}^{\mathcal{H}}(G)$ is indeed necessary and it is not sufficient that MSO-MC$_\phi$ is polynomial-time tractable on $\mathcal{H}$. Condition 2 is also necessary for our approach to work, as we need some (approximate) well-structured modulator; luckily, Section 12.2 shows that a wide variety of studied graph classes satisfy this condition. Finally, it is easy to see that Theorem 11.19 rules out an extension of Theorem 12.23 to MSO-Opt problems (which was possible in the special case considered in Section 12.1). We provide in the following lemma another example of an MSO-Opt problem that is NP-hard on the class of graphs with $\mathrm{wsn}_1^{\mathcal{H}} \leq 1$.

Lemma 12.24. *There exists an* MSO_1 *formula* φ *and a graph class* $\mathcal{H}$ *characterized by a finite obstruction set such that* MSO-Opt$_{\varphi}^{\leq}$ *admits a bikernel parameterized by* $\mathrm{mod}^{\mathcal{H}}$ *but is* NP*-hard on the class of graphs with* $\mathrm{wsn}_1^{\mathcal{H}} \leq 1$.

Proof. Consider the formula $\varphi(S) = \mathrm{fvs}(S) \vee \deg(S)$, where $\mathrm{fvs}(S)$ expresses that S is a feedback vertex set in G and $\deg(S)$ expresses that S is a modulator to graphs with maximum degree 4. Let $\mathcal{H}$ be the class of graphs of maximum degree 4.

First, we prove that MSO-Opt$_{\varphi}^{\leq}$ admits a polynomial bikernel parameterized by $k = \mathrm{mod}^{\mathcal{H}}(G)$. Consider the following algorithm: given an instance (G, r) of MSO-Opt$_{\varphi}^{\leq}$, we check if $r \geq k$. If this is the case, we can immediately output YES, since k is the minimum size of a set S satisfying $\deg(S)$. If $r < k$, we compute a polynomial kernel (G_1, r_1) of MSO-Opt$_{\mathrm{fvs}}^{\leq}$ from (G, r), and a polynomial kernel (G_2, r_2) for MSO-Opt$_{\deg}^{\leq}$ from (G, r). Both (G_1, r_1) and (G_2, r_2) have size bounded by a polynomial of k, G_1 can be constructed by using any kernelization algorithm for Feedback Vertex Set [193], and G_2 can be constructed by enumerating all obstructions (supergraphs of $K_{1,5}$) and then using a kernelization algorithm for 6-Hitting Set [2]. Now it is easily observed that (G, r) is a YES instance of MSO-Opt$_{\varphi}^{\leq}$ if and only if $((G_1, r_1), (G_2, r_2))$ is a YES instance of the following problem: given an instance (A, a) of Feedback Vertex Set and an instance (B, b) of 6-Hitting Set, answer YES if and only if at least one of (A, a) and (B, b) is a YES instance.

To conclude the proof, we show that MSO-Opt$_{\varphi}^{\leq}$ is NP-hard even when G has a $(1, 1)$-well-structured modulator to $\mathcal{H}$. We reduce from the (unparameterized) Feedback Vertex Set problem on graphs of degree at most 4, which is known to be NP-hard [98]. Let (G, r) be an instance of Feedback Vertex Set where G is an n-vertex graph of degree at most 4. We construct the graph G' from G by adding a single vertex x and a disjoint union of $n + 1$ stars with five leaves, denoted $S_1, \dots, S_{n+1}$, and then adding an edge from x to the center of each star and from x to a single arbitrary vertex in G. Observe that any feedback vertex set in G is also a feedback vertex set in G', and that any set S sayisfying $\deg(S)$ must have cardinality at least $n + 1$. Hence (G, r) is a YES-instance of Feedback Vertex Set if and only if (G', r) is a YES-instance of MSO-Opt$_{\varphi}^{\leq}$. Since x together with the stars added in G' forms a tree (which has rank-width 1), it follows that this forms a $(1, 1)$-well-structured modulator to $\mathcal{H}$. □

12.3.1 Applications of Theorem 12.23

As our first general application, we consider the results of Gajarský et al. [92]. Their main result is summarized below.

Fact 12.25 ([92])**.** *Let Π be a problem with finite integer index, $\mathcal{K}$ a class of graphs of bounded expansion, $d \in \mathbb{N}$, and $\mathcal{H}$ be the class of graphs of treedepth at most d. Then there exists an algorithm that takes as input $(G, \xi) \in \mathcal{K} \times \mathbb{N}$ and in time $\mathcal{O}(|G| + \log \xi)$ outputs (G', ξ') such that*

1. *$(G, \xi) \in \Pi$ if and only if $(G', \xi') \in \Pi$;*
2. *G' is an induced subgraph of G; and*
3. *$|G'| = \mathcal{O}(\mathrm{mod}^{\mathcal{H}}(G))$.*

The following fact provides a link between the notion of finite integer index used in the above result and the MSO-MC$_\varphi$ problems considered in this part.

Fact 12.26 ([11], see also [26])**.** *For every MSO_1 sentence φ, it holds that* MSO-MC$_\varphi$ *is finite-state and hence has finite integer index.*

Theorem 12.27. *Let $c, d \in \mathbb{N}$ and $\mathcal{H}$ be the class of graphs of treedepth at most d. For every MSO_2 sentence φ, it holds that* MSO-MC$_\varphi$ *admits a linear kernel parameterized by $\mathrm{wsn}_c^{\mathcal{H}}$ on any class of graphs of bounded expansion.*

Proof. Since G belongs to some fixed class of bounded expansion, it follows by Observation 12.12 that G is uniformly ℓ-sparse for some constant ℓ and by Fact 12.10 we can translate ψ to an equivalent MSO_1 sentence. Our proof then relies on Theorem 12.23. From Fact 12.26 and Fact 12.25 it follows that MSO-MC$_\psi$ admits a kernel of size $\mathcal{O}(\mathrm{mod}^{\mathcal{H}}(G))$. Hence the assumptions of the theorem satisfy Condition 1 of Theorem 12.23, where the polynomial p is a linear function.

By Fact 3.6 $\mathcal{H}$ can be characterized by finite set of forbidden induced subgraphs. Therefore, by Theorem 12.22 there exists a polynomial algorithm that can find a (k, c)-well-structured modulator to $\mathcal{H}$ and $k \leq r \cdot \mathrm{wsn}_c^{\mathcal{H}}$, where $r \in \mathbb{N}$ is a constant (depending on d). Therefore, Condition 2 of Theorem 12.23 is also satisfied, where the polynomial q is also linear function. Hence we conclude that MSO-MC$_\psi$ and hence admits a kernel of size $r \cdot \mathrm{wsn}_c^{\mathcal{H}}(G)$. □

As our second general application, we consider well-structured modulators to the class of forests. Lemma 12.29 shows that feedback vertex set may be used to kernelize any MSO-definable decision problem on graphs of bounded degree. However, before we proceed to the lemma itself, we need to briefly introduce the protrusion replacement rule.

Given a graph G, an r-protrusion is a set of vertices $L \subseteq V(G)$ such that $|N(G - L)| \leq r$ and the treewidth of $G[L]$ is at most r. The set $N(G - L)$ is called the boundary of L. A much more in-depth explanation of protrusion replacement can be found, e.g., in [26] or [138].

Fact 12.28 ([26, 138]). *Let $r \in \mathbb{N}$ be a constant, ϕ be a fixed MSO_2 formula and G be a graph containing an r-protrusion L. Then there exists a constant-size graph G'_L such that the graph G' obtained by deleting L and making at most r vertices of G'_L adjacent to the neighbors of L satisfies $G' \models \varphi$ iff $G \models \varphi$. Furthermore the graph G'_L may be computed from G and L in polynomial time.*

The graph G'_L is sometimes called a representative; we remark that the algorithm above computing G'_L is constructive, but contains huge constants. We note that the following lemma could also be obtained by a straightforward application of a result by Kim et al. [138]. However, we provide an alternative proof of this lemma for the completeness.

Lemma 12.29. *Let $\mathcal{F}$ be the class of forests and $d \in \mathbb{N}$. For every MSO_2 sentence φ, it holds that* MSO-MC$_\varphi$ *admits a linear kernel parameterized by* $\mathrm{mod}^{\mathcal{F}}$ *on every class of graphs of degree at most d.*

Proof. Let G' be the input graph. We first use Fact 12.18 to compute a feedback vertex set $X \subseteq V(G)$ of cardinality $k \leq 2\mathrm{mod}^{\mathcal{F}}(G)$. Let G'_0 denote the graph containing all the connected components of $G' - X$ which are not adjacent to X. Since G'_0 has treewidth 1 and an empty boundary, by Fact 12.28 we can find a constant-size graph G^*_0 such that the graph G^* obtained from G' by replacing G'_0 with G^*_0 satisfies the following: $G^* \models \varphi$ iff $G' \models \varphi$.

Let G be obtained from G^* by deleting G^*_0. Then the number of connected components in $H = G - X$ is bounded by $k \cdot d$, since each tree in H is adjacent to at least one vertex in X. We proceed by *marking* each vertex in H which is adjacent to some vertex in X; note that this process marks at most dk vertices. This is followed by a secondary marking procedure, where we mark each vertex $v \in V(H)$ of degree at least 3 with the following property: deleting v separates its connected component in H into at least 3 connected components of H, each containing at least 1 marked vertex. We argue that this secondary marking procedure does not mark more than dk vertices as well. Observe that if we remove all non-marked leaves of H, then we would mark a superset of vertices marked by the secondary mark procedure. Therefore, for the sake of this argument, we can consider a subgraph of H, with all leaves marked that preserve reachability of marked vertices. Since, the number of vertices of degree 3 is at most the number of leaves in a forest, it follows that the secondary marking procedure marks at most dk vertices and the total number of marked vertices is bounded by $2dk$. Let us denote the set of all marked vertices M.

Let u be the order of the largest representative G'_L as per Fact 12.28 for treewidth 1, boundary-size 2 and φ. Now assume that H contains a tree T with z marked vertices. Observe, that $T - M$ has at most $z \cdot d$ connected components. Moreover, every connected component of $T - M$ is adjacent with at most 2 marked vertices, otherwise a vertex inside this component would be marked. Let C be a connected component in $T - M$ and let T_C be the subtree of T containing precisely C and the, at most 2, marked vertices adjacent to C. Clearly, T_C forms a subgraph in G with a boundary size of 2 and treewidth 1. Therefore, if $|T_C| > u$, then by Fact 12.28 we can in polynomial time replace T_C by a smaller representative of T_C and obtain an equivalent instance. The step outlined in this paragraph takes polynomial time and is guaranteed to reduce the order of G by at least 1.

It follows that if the replacement procedure is not applicable, then T has at most $z + z \cdot d \cdot u = (d \cdot u + 1)z$ vertices. Since the total number of marked vertices in H is at most $2dk$, it follows that $V(H) \in \mathcal{O}(k)$. In particular, this implies that $V(G^*) \in \mathcal{O}(k)$, which means that we have a linear kernel. □

With Lemma 12.29, the proof of the following theorem is analogous to the proof of Theorem 12.27.

Theorem 12.30. *Let $c \in \mathbb{N}$ and $\mathcal{F}$ be the class of forests. For every MSO_2 sentence φ, it holds that* MSO-MC$_\varphi$ *admits a linear kernel parameterized by* $\text{wsn}_c^{\mathcal{F}}$ *on every class of graphs of bounded degree.*

Proof. Since the degree of the input graph G is bounded by some constant d, it follows that G is uniformly $\frac{d}{2}$-sparse. The theorem now once again follows from Theorem 12.23. In particular, Condition 1 follows directly from Lemma 12.29 with a linear function p. Theorem 12.19 then gives us a polynomial time algorithm to find a 3-approximation of $\text{wsn}_c^{\mathcal{F}}$, satisfying Condition 2 with q also linear. Therefore we conclude by Theorem 12.23 that MSO-MC$_\varphi$ admits a linear kernel parameterized by $\text{wsn}_c^{\mathcal{H}}$ on any class of graphs of bounded degree. □

12.4 Summary and Open Questions

Our results in last two chapters show that measuring the structure of modulators can lead to an interesting and, as of yet, relatively unexplored spectrum of structural parameters. Such parameters have the potential of combining the best of decomposition-based techniques and modulator-based techniques, and can be applied both in the context of kernelization and FPT algorithms. We believe that further work in the direction of modulators will allow us to push the frontiers of tractability towards new, uncharted classes of inputs.

One possible direction for future research is the question of whether the class of MSO_1-definable problems considered in Theorem 12.23 can be extended to other finite-state problems. It would of course also be interesting to see more applications of Theorem 12.23 and new methods for approximating well-structured modulators. Last but not least, we mention that the split-modules used in the definition of our parameters could in principle be refined to less restrictive notions (for instance cuts of constant *cut-rank* [168]); such a relaxed parameter could still be used to obtain polynomial kernels, as long as there is a way of efficiently approximating or computing such modulators.

Notes

The results in this chapter appeared in a conference paper in the proceedings of 10th International Symposium on Parameterized and Exact Computation (IPEC 2015) [73]. The results in this chapter will also appear in a journal publication in a special issue of Discrete Applied Mathematics.

Part IV

Conclusions

CHAPTER 13

Summary

In this chapter we summarize our results. In this thesis we explored possibilities how structure can help us solve NP-hard problems more efficiently. To mathematically capture the structure of input instance, we used the paradigm of parameterized complexity. We established several fixed-parameter tractability results for a variety of problems exploiting already established as well as new kinds of structural parameters. More precisely, we studied two different approaches to design of structural parameters, decompositions and modulators, and we developed a novel combined approach that allows us to push the boundaries of tractability for many problems.

13.1 Decomposition Parameters

We began by an investigation of decomposition parameters.

13.1.1 Counting Linear Extensions

Here, we first establish parameterized complexity of #LinExt parameterized by the treewidth of two different graph representations of a poset.

- We showed that there is no FPT algorithm for solving #LinExt parameterized by the treewidth of the cover graph, unless FPT = W[1].
- We gave an FPT algorithm for #LinExt, where parameter is the treewidth of incomparability graph.

13.1.2 Decomposition Parameters for QBF

Afterwards, we switched our focus to the archetypal PSPACE-complete problem QBF. QBF is know to remain PSPACE-complete even when restricted to the instances with the pathwidth of the primal graph at most 4. However, we showed that if the path decomposition is forced to abide to the restrictions imposed by dependencies between the variables, we can still use such path decompositions to efficiently solve QBF instances.

- We introduced the parameter prefix pathwidth, which is an extension of the pathwidth that takes into account not only the structure of clauses in the formula, but also dependencies between the variables given by some dependency scheme.
- We showed that we can use a prefix path decomposition of width bounded by the parameter k to solve the given QBF instance in FPT time.
- Finally, we develop several FPT and polynomial time approximation algorithms for computing prefix path decomposition of small width for given combination of QBF formula and dependency scheme.

13.2 Modulators

In the second part, we focused on the investigation of algorithmic applications of modulators for various problems.

13.2.1 Modulators to Distance-Hereditary Graphs

We started by the study of modulators to distance-hereditary graphs, which is a well-studied graph class that is especially important because it is exactly the class of graphs with rank-width 1.

- We presented the first single-exponential FPT algorithm for vertex deletion to distance-hereditary graphs. This is particularly important, as it is the first single-exponential FPT algorithm for a modulator to a "full" class of graphs of bounded rank-width.
- We complemented our result with matching asymptotic lower bounds based on ETH.
- We showed that we can use modulators to distance-hereditary graphs as a parameter for solving classical NP-hard problems in a single-exponential FPT time.

13.2.2 Towards a Polynomial Kernel for Directed Feedback Vertex Set

The second problem we studied in this part is DFVS. It is a long standing open question in parameterized complexity whether DFVS admits a polynomial kernel parameterized by the solution size. Our results provide a stepping stone towards resolving the existence of a polynomial kernel for DFVS, and to the best of our knowledge also represent the first kernelization results for DFVS with respect to any natural parameter.

- We gave a kernel with $\mathcal{O}(k^4)$ vertices for DFVS parameterized by the size of the feedback vertex set of the underlying undirected graph.
- Moreover, we gave a kernel with $\mathcal{O}(k)$ vertices for DFVS with the same parameter, when the input is embeddable on a surface of constant genus.

13.2.3 Modulators for ILPs

The last problem we examine in the modulators part is ILP. In order to overcome the complexity barriers of ILP, a wide range of problems have been encoded in restricted variants of ILP such as 2-stage stochastic ILP and N-fold ILP; examples for the former include a range of transportation and logistic problems [173, 120], while examples for the latter range from scheduling [142] to, e.g., computational social choice [143].

- We introduced a novel parameter fracture modulator, which provides a unified platform which generalizes 2-stage stochastic ILP, N-fold ILP and also 4-block N-fold ILP.
- We identified and analyzed tree separate cases depending on whether we allow only global variables, only global constrains, or both.
- We identified under which circumstances we can use these parameters to obtain FPT and XP algorithms.
- One may view our algorithmic results as "algorithmic meta-theorems" for ILP, where previously known algorithms for 2-stage stochastic ILP, N-fold ILP, and 4-block N-fold ILP only represent a simple base case.

13.3 Hybrid Parameters

In the last technical part of the thesis, we concentrated our attention towards combining the two approaches to a single hybrid approach. Here we presented a family of parameters, which are based on the following idea. Similarly as in the modulator approach, for an input graph G, we are looking for vertex set S, such that $G \setminus S$ belongs to some tractable class $\mathcal{H}$. However, instead of parameterizing by the size of a smallest modulator to $\mathcal{H}$, we measure its structure by some decomposition parameter and consequently, we parameterize by the minimum value of this decomposition parameter over all modulators to $\mathcal{H}$ in G.

13.3.1 Well-Structured Modulators

- We introduced (k, c)-well-structured modulators to a graph class $\mathcal{H}$, which are basically modulators that consist of k disjoint split-modules each of rank-width at most c.

13.3.2 FPT Algorithms for Well-Structured Modulators

Afterwards, we studied algorithmic applications of well-structured modulators. We started by the investigation of FPT algorithms using well-structured modulators.

- We developed FPT algorithms for computing $\mathrm{wsn}^{\mathcal{H}}$ for many graph classes $\mathcal{H}$. In particular, for $\mathcal{H}$ being the class of forests, chordal graphs, or any class characterized by a finite set of forbidden induced subgraphs.
- We designed FPT algorithms for Vertex Cover and Clique parameterized by $\mathrm{wsn}^{\mathcal{H}}$, for any class $\mathcal{H}$, such that 1) the given problem is solvable in polynomial time on $\mathcal{H}$ and 2) k-well-structured modulator to $\mathcal{H}$ can be approximated in FPT time.
- We developed a meta-theorem to obtain FPT algorithms for problems definable in MSO_1 logic parameterized by $\mathrm{wsn}^{\mathcal{H}}$.
- We showed that, in general, solving MSO-Opt problems [50, 93] is not FPT when parameterized by $\mathrm{wsn}^{\mathcal{H}}$.

13.3.3 Meta-Kernelization using Well-Structured Modulators

Finally, in the last chapter of the third part we studied the possibility of using well-structured modulators for obtaining polynomial kernels. We showed that for a fixed constant c, (k, c)-well-structured modulators can be used to obtain kernels for a plethora of problems.

- We developed polynomial time approximation algorithms for computing $\mathrm{wsn}^{\mathcal{H}}$ for various graph classes $\mathcal{H}$.
- We developed a meta-theorem to obtain kernels for problems definable in MSO_1 logic parameterized by $\mathrm{wsn}^{\mathcal{H}}$.
- We showed that our results subsume the framework of Ganian, Slivovsky, and Szeider [97] and lift the framework by Gajarský et al. [92] to more general graph classes.

CHAPTER 14

Open Problems and Future Directions

The theoretical investigation that we performed in this thesis leaves a number of open questions and brings forth plenty of topics for future research. In this chapter, we first briefly summarize open questions for specific problems we studied. Afterwards, we give some possible directions for developing new kinds of structure that can be exploited for obtaining FPT algorithms.

14.1 Summary of Open Questions for Individual Problems

Counting Linear Extensions

The parameterized complexity of #LinExt under other very natural parameterizations such as the width of the poset or the treewidth of the poset graph is still open. Moreover, our intractability result for the treewidth of the cover graph gives rise to the question for whether the problems becomes tractable for some stronger parameterizations such as the treewidth of the poset graph, the treedepth or even vertex cover number of the poset- or cover graph, or a combinations of these parameters with other natural parameters for posets such as the width, the dimension, or the height of the poset. As a side note it would also be interesting to establish whether our hardness result for #LinExt can be sharpened to #W[1]-completeness [85] and to obtain matching membership results.

Finally, we think that techniques employed in our hardness result might be valuable for other counting problems. It would be interesting to see if we can combine fpt turing reductions with counting modulo some (prime) number to obtain similar hardness results for other parameterized counting problems.

Prefix Pathwidth for QBF

Our results push the frontiers of tractability for QBF to new natural classes of instances. A number of interesting research questions still remain open in the area, and perhaps the most prominent of these is whether one can lift our results towards prefix treewidth. This would be especially interesting for posets of unbounded width, since on bounded-width posets these parameters differ only by a constant factor. The exact complexity of computing prefix treewidth and prefix pathwidth on general posets remains a challenging open problem.

Distance-Hereditary Vertex Deletion

As distance-hereditary graphs are exactly graphs of rankwith 1, our algorithm represents a first step in resolving the following prominent open question. "Does Rank-width-c Vertex Deletion admit a single-exponential FPT algorithm parameterized by the solution size?". Moreover, the existence of a polynomial kernel or an approximation algorithm for such vertex deletion problems for $c > 1$ remains open as well.

Directed Feedback Vertex Set

A possible direction for future work is to ask if we can find reasonable parameters "between" DFVS number and FVS number, for which we can generalize our polynomial kernel. Similarly, one can study also other parameters that are incomparable to the FVS number but also upper-bound the DFVS. Regarding our result for bounded genus graphs, one can ask if the linear kernel can be lifted to other sparse graph classes such as graph classes of bounded expansion or nowhere dense graphs?

Another related problem of interest is whether DFVS can be solved in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$, which remains open even on planar graphs.

Fractured Modulators for ILP

The only remaining blank part in the complexity map that we presented in Chapter 9 is the question of whether mixed fracture modulators admit a fixed-parameter algorithm in case of bounded coefficients; we believe that this is in fact a major open problem in the area. A first step towards settling this question would be to resolve the fixed-parameter (in)tractability of 4-block N-fold ILP, which was also left open in previous work; progress in this direction seems to require new techniques and insights [116].

Well-Structured Modulators

We believe that further work in the direction of structured modulators will allow us to push the frontiers of tractability towards new, uncharted classes of inputs. One possible direction for future research is the question of whether the results for MSO_1-definable problems can be extended to other finite-state problems. It would also be interesting to see more applications for our meta-theorems as well as new algorithms for computing well-structured modulators to new graph classes. Last but not least, a split-module can be seen as a subset X of vertices of G such that the adjacency matrix of the bipartite subgraphs of G induced by the bipartition X and $V(G) \setminus S$ has rank 1. In principle this could be refined to obtain a less restrictive notions of split-modules; using our techniques it is possible to obtain similar meta-theorems for using the well-structured modulators even when the cut-rank [168] between the split-module and the rest of the graph is some fixed constant instead of 1. However, it is not clear whether we can find, or at least approximate, such more general well-structured modulators efficiently.

14.2 General Directions for Future work

Tweaking Standard Structural Parameters

We have seen in Chapter 5, that even though QBF remains PSPACE-complete for instances with pathwidth of the primal graph at most 4, it is possible to obtain an FPT algorithms if we enrich the path decomposition in a way that it is also required to abide to restrictions imposed by dependencies between variables. It would be interesting to see, if we can obtain similar results for other problems that are known to be fixed-parameter intractable with respect to established structural parameters. Especially, we would like to concentrate on high-impact problems emerging from the areas such as Artificial Intelligence or Reasoning.

Modulators with a Certain Structure

In Part III, we introduced well-structured modulators to some fixed graph class, where we measured the structure of modulator as the minimum number of split-modules with bounded rank-width. However, in general, one can define any measure that describes how structured the modulator is. This leads us to the question of identifying algorithmically useful measures τ and finding modulators of small τ efficiently. For such measure to be useful, we should be both able to find modulators with small measure efficiently and once provided a modulator of small measure, we should be able to use it obtain efficient algorithm.

Another possible direction for future work is to study the structured modulators to Satisfiability and Reasoning. Modulators in the setting of Satisfiability and other reasoning problems such as CSP or QBF were studied under the notion of *backdoors* (see, e.g., [183, 100, 99]). Structured modulators translate naturally to backdoor sets, and again the fundamental tractable classes of SAT, Horn and 2CNF, form natural base classes for an initial study. Moreover, there are multiple ways how to assign a graph to an instance of various reasoning problems and study structured backdoor sets. For example, previous work shows that the modular structure of variables in the so-called *signed incidence graph* can be used to speed up SAT decision [94, 170], and we expect interesting results in the new context of structured backdoors.

A further promising direction is to study what we call *recursive modulators*. Roughly speaking, a recursive modulator is a vertex set that decomposes the graph into several tractable regions. We use an instance of Independent Set to showcase one potential application of our approach. Consider an input graph with the structure depicted by Figure 14.1. Independent Set is known to be solvable in polynomial time, e.g., on

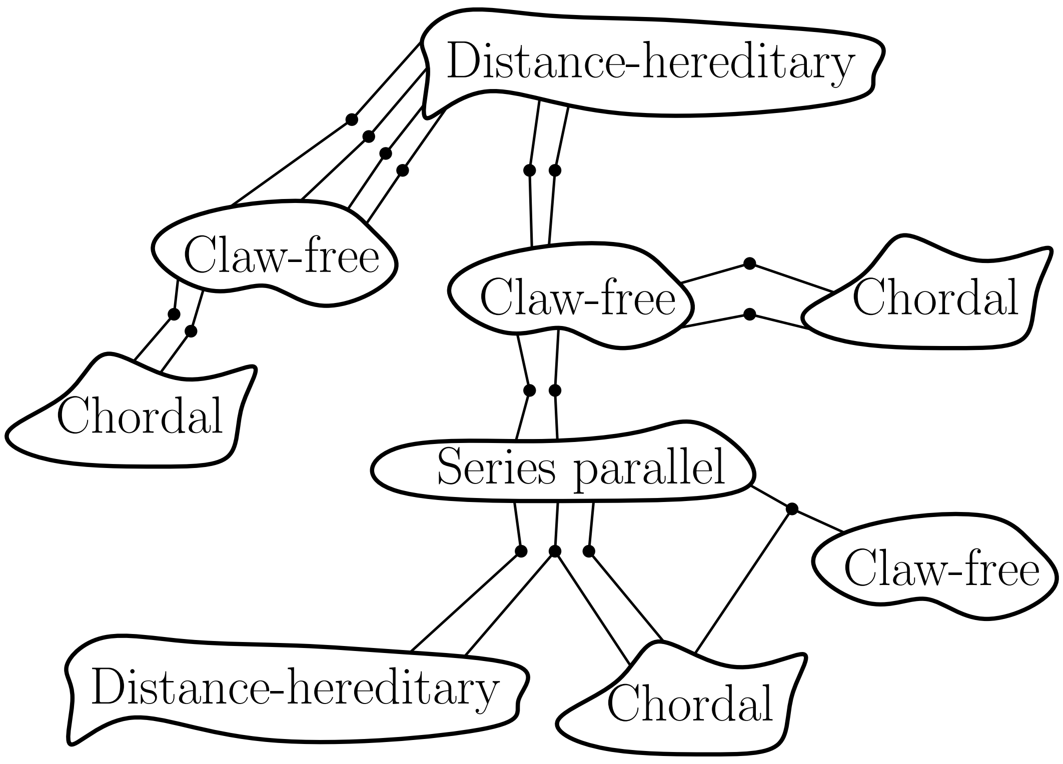


Figure 14.1: An example of recursive modulator.

series parallel graphs [23], claw-free graphs [155], distance-hereditary graphs [50] and chordal graphs [164]. But the input graph need not contain a small modulator to any of these classes—in fact, even the modulator depicted in our example may be large in general. Yet, if the modulator decomposes the graph into a tree-like structure, and the neighborhood of each component outside of the modulator is bounded by, for instance, $k = 6$, then we may apply recursive techniques on such a decomposition-like structure to solve the whole instance in FPT time parameterized by k. In particular, it is possible to apply an FPT leaf-to-root dynamic programming algorithm which solves the instance as long as such a decomposition is provided.

Index

Bibliography

[1] K. R. Abrahamson, R. G. Downey, and M. R. Fellows. Fixed-parameter intractability II (extended abstract). In P. Enjalbert, A. Finkel, and K. W. Wagner, editors, *STACS 93, 10th Annual Symposium on Theoretical Aspects of Computer Science, Würzburg, Germany, February 25-27, 1993, Proceedings*, volume 665 of *Lecture Notes in Computer Science*, pages 374–385. Springer, 1993.

[2] F. N. Abu-Khzam. A kernelization algorithm for d-hitting set. *J. Computer and System Sciences*, 76(7):524–531, 2010.

[3] I. Adler, M. M. Kanté, and O. Kwon. Linear rank-width of distance-hereditary graphs i. A polynomial-time algorithm. *Algorithmica*, 78(1):342–377, 2017.

[4] I. Adler and M. Weyer. Tree-width for first order formulae. *Logical Methods in Computer Science*, 8(1), 2012.

[5] A. Agrawal, S. Kolay, D. Lokshtanov, and S. Saurabh. A faster FPT algorithm and a smaller kernel for block graph vertex deletion. In E. Kranakis, G. Navarro, and E. Chávez, editors, *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, volume 9644 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2016.

[6] M. Aigner and G. M. Ziegler. *Proofs from THE BOOK.* Springer, 5th edition, 2014.

[7] V. E. Alekseev. Polynomial algorithm for finding the largest independent sets in graphs without forks. *Discrete Applied Mathematics*, 135(1-3):3–16, 2004.

[8] V. E. Alekseev, D. V. Korobitsyn, and V. V. Lozin. Boundary classes of graphs for the dominating set problem. *Discrete Mathematics*, 285(1-3):1–6, 2004.

[9] N. Alon, G. Gutin, E. J. Kim, S. Szeider, and A. Yeo. Solving max-r-sat above a tight lower bound. *Algorithmica*, 61(3):638–655, 2011.

[10] S. A. Alumur and B. Y. Kara. Network hub location problems: The state of the art. *European Journal of Operational Research*, 190(1):1–21, 2008.

[11] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. *J. ACM*, 40(5):1134–1164, 1993.

[12] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

[13] S. Arora and B. Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[14] M. D. Atkinson. On computing the number of linear extensions of a tree. *Order*, 7(1):23–25, 1990.

[15] A. Atserias and S. Oliva. Bounded-width QBF is PSPACE-complete. *J. Computer and System Sciences*, 80(7):1415–1429, 2014.

[16] V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discrete Mathematics*, 12(3):289–297, 1999.

[17] H. Bandelt and H. M. Mulder. Distance-hereditary graphs. *J. Combinatorial Theory, Series B*, 41(2):182–208, 1986.

[18] J. Bang-Jensen and G. Gutin. *Digraphs - theory, algorithms and applications.* Springer, 2002.

[19] J. Barát. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22(2):161–172, 2006.

[20] A. Becker and D. Geiger. Optimization of pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artificial Intelligence*, 83(1):167–188, 1996.

[21] B. Bergougnoux, E. Eiben, R. Ganian, S. Ordyniak, and M. S. Ramanujan. Towards a polynomial kernel for directed feedback vertex set. In K. G. Larsen, H. L. Bodlaender, and J. Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, volume 83 of *LIPIcs*, pages 36:1–36:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[22] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Computing*, 25(6):1305–1317, 1996.

[23] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998.

[24] H. L. Bodlaender, J. S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller, and Z. Tuza. Rankings of graphs. *SIAM J. Discrete Mathematics*, 11(1):168–181, 1998.

[25] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Computer and System Sciences*, 75(8):423–434, 2009.

[26] H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos. (meta) kernelization. *J. ACM*, 63(5):44:1–44:69, 2016.

[27] H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Kernel bounds for path and cycle problems. *Theoretical Computer Science*, 511:117–136, 2013.

[28] H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Preprocessing for treewidth: A combinatorial analysis through kernelization. *SIAM J. Discrete Mathematics*, 27(4):2108–2142, 2013.

[29] H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Mathematics*, 28(1):277–305, 2014.

[30] H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

[31] A. Bouchet. Transforming trees by successive local complementations. *J. Graph Theory*, 12(2):195–207, 1988.

[32] A. Brandstädt and V. V. Lozin. A note on alpha-redundant vertices in graphs. *Discrete Applied Mathematics*, 108(3):301–308, 2001.

[33] B. Bresar, M. Changat, S. Klavzar, M. Kovse, J. Mathews, and A. Mathews. Cover-incomparability graphs of posets. *Order*, 25(4):335–347, 2008.

[34] G. R. Brightwell and P. Winkler. Counting linear extensions is #P-complete. In C. Koutsougeras and J. S. Vitter, editors, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 175–181. ACM, 1991.

[35] R. Bubley and M. E. Dyer. Faster random generation of linear extensions. *Discrete Mathematics*, 201(1-3):81–88, 1999.

[36] L. Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003.

[37] L. Cai, J. Chen, R. G. Downey, and M. R. Fellows. On the structure of parameterized problems in NP (extended abstract). In P. Enjalbert, E. W. Mayr, and K. W. Wagner, editors, *STACS 94, 11th Annual Symposium on Theoretical Aspects of Computer Science, Caen, France, February 24-26, 1994, Proceedings*, volume 775 of *Lecture Notes in Computer Science*, pages 509–520. Springer, 1994.

[38] L. Cai and D. W. Juedes. On the existence of subexponential parameterized algorithms. *J. Computer and System Sciences*, 67(4):789–807, 2003.

[39] Y. Cao, J. Chen, and Y. Liu. On feedback vertex set: New measure and new structures. *Algorithmica*, 73(1):63–86, 2015.

[40] Y. Cao and D. Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016.

[41] H. Chen and V. Dalmau. Decomposing quantified conjunctive (or disjunctive) formulas. *SIAM J. Computing*, 45(6):2066–2086, 2016.

[42] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, 2008.

[43] S. Cicerone and G. D. Stefano. Graph classes between parity and distance-hereditary graphs. *Discrete Applied Mathematics*, 95(1-3):197–216, 1999.

[44] O. Cogis and E. Thierry. Computing maximum stable sets for distance-hereditary graphs. *Discrete Optimization*, 2(2):185–188, 2005.

[45] S. A. Cook. The complexity of theorem-proving procedures. In M. A. Harrison, R. B. Banerji, and J. D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971.

[46] B. Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[47] B. Courcelle. The monadic second-order logic of graphs XIV: uniformly sparse graphs and edge set quantifications. *Theoretical Computer Science*, 299(1-3):1–36, 2003.

[48] B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

[49] B. Courcelle, J. Engelfriet, and G. Rozenberg. Context-free handle-rewriting hypergraph grammars. In H. Ehrig, H. Kreowski, and G. Rozenberg, editors, *Graph-Grammars and Their Application to Computer Science, 4th International Workshop, Bremen, Germany, March 5-9, 1990, Proceedings*, volume 532 of *Lecture Notes in Computer Science*, pages 253–268. Springer, 1990.

[50] B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

[51] W. H. Cunningham. Decomposition of directed graphs. *SIAM J. Algebraic Discrete Methods*, 3(2):214–228, 1982.

[52] W. H. Cunningham and J. Edmonds. A combinatorial decomposition theory. *Canadian Journal of Mathematics*, 32:734–765, 1980.

[53] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms.* Springer, 2015.

[54] E. Dahlhaus. Parallel algorithms for hierarchical clustering and applications to split decomposition and parity graph recognition. *J. Algorithms*, 36(2):205–240, 2000.

[55] P. Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theoretical Computer Science*, 351(3):337–350, 2006.

[56] A. D'Atri and M. Moscarini. Distance-hereditary graphs, steiner trees, and connected domination. *SIAM J. Computing*, 17(3):521–538, 1988.

[57] J. A. De Loera, R. Hemmecke, and M. Köppe. *Algebraic and Geometric Ideas in the Theory of Discrete Optimization*, volume 14 of *MOS-SIAM Series on Optimization.* SIAM, 2013.

[58] R. Diestel. Graph minors 1: A short proof of the path-width theorem. *Combinatorics, Probability & Computing*, 4:27–30, 1995.

[59] R. Diestel. *Graph Theory*, volume 173 of *Graduate texts in mathematics.* Springer, 4th edition, 2012.

[60] R. G. Downey and M. R. Fellows. Fixed-parameter intractability. In *Proceedings of the Seventh Annual Structure in Complexity Theory Conference, Boston, Massachusetts, USA, June 22-25, 1992*, pages 36–49. IEEE Computer Society, 1992.

[61] R. G. Downey and M. R. Fellows. Fixed parameter tractability and completeness. In K. Ambos-Spies, S. Homer, and U. Schöning, editors, *Complexity Theory: Current Research, Dagstuhl Workshop, February 2-8, 1992*, pages 191–225. Cambridge University Press, 1992.

[62] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Computing*, 24(4):873–921, 1995.

[63] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer, 1999.

[64] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.

[65] R. G. Downey, M. R. Fellows, and U. Taylor. The parameterized complexity of relational database queries and an improved characterization of W[1]. In D. S. Bridges, C. S. Calude, J. Gibbons, S. Reeves, and I. H. Witten, editors, *First*

Conference of the Centre for Discrete Mathematics and Theoretical Computer Science, DMTCS 1996, Auckland, New Zealand, December, 9-13, 1996, pages 194–213. Springer-Verlag, Singapore, 1996.

[66] P. G. Drange, M. S. Dregi, and P. van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016.

[67] P. Dvorák, E. Eiben, R. Ganian, D. Knop, and S. Ordyniak. Solving integer linear programs with a small number of global variables and constraints. In C. Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 607–613. ijcai.org, 2017.

[68] M. E. Dyer, A. M. Frieze, and R. Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991.

[69] U. Egly, T. Eiter, H. Tompits, and S. Woltran. Solving advanced reasoning tasks using quantified boolean formulas. In H. A. Kautz and B. W. Porter, editors, *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, pages 417–422. AAAI Press / The MIT Press, 2000.

[70] E. Eiben, R. Ganian, K. Kangas, and S. Ordyniak. Counting linear extensions: Parameterizations by treewidth. In P. Sankowski and C. D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 39:1–39:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[71] E. Eiben, R. Ganian, and O. Kwon. A single-exponential fixed-parameter algorithm for distance-hereditary vertex deletion. In P. Faliszewski, A. Muscholl, and R. Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, volume 58 of *LIPIcs*, pages 34:1–34:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[72] E. Eiben, R. Ganian, and S. Ordyniak. Using decomposition-parameters for QBF: mind the prefix! In D. Schuurmans and M. P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 964–970. AAAI Press, 2016.

[73] E. Eiben, R. Ganian, and S. Szeider. Meta-kernelization using well-structured modulators. In T. Husfeldt and I. A. Kanj, editors, *Parameterized and Exact Computation - 10th International Symposium, IPEC 2014, Patras, Greece, September 16-18, 2015. Revised Selected Papers*, volume 43 of *LIPIcs*, pages 114–126. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[74] E. Eiben, R. Ganian, and S. Szeider. Solving problems on graphs of high rank-width. In F. Dehne, J. Sack, and U. Stege, editors, *Algorithms and Data Structures Symposium (WADS 2015), August 5-7, 2015, University of Victoria, BC, Canada*, LNCS, pages 314–326. Springer, 2015.

[75] E. Eiben, R. Ganian, and S. Szeider. Solving problems on graphs of high rank-width. *Algorithmica*, 80(2):742–771, 2018.

[76] P. Erdös and P. Turán. On a problem of Sidon in additive number theory, and on some related problems. *J. the London Mathematical Society*, 1(4):212–215, 1941.

[77] M. R. Fellows. The lost continent of polynomial time: Preprocessing and kernelization. In H. L. Bodlaender and M. A. Langston, editors, *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, volume 4169 of *Lecture Notes in Computer Science*, pages 276–277. Springer, 2006.

[78] M. R. Fellows, F. V. Fomin, D. Lokshtanov, F. A. Rosamond, S. Saurabh, S. Szeider, and C. Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Information and Computation*, 209(2):143–153, 2011.

[79] M. R. Fellows, D. Lokshtanov, N. Misra, F. A. Rosamond, and S. Saurabh. Graph layout problems parameterized by vertex cover. In S. Hong, H. Nagamochi, and T. Fukunaga, editors, *Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings*, volume 5369 of *Lecture Notes in Computer Science*, pages 294–305. Springer, 2008.

[80] S. Felsner and T. Manneville. Linear extensions of N-free orders. *Order*, 32(2):147–155, 2014.

[81] S. Felsner, V. Raghavan, and J. Spinrad. Recognition algorithms for orders of small width and graphs of small dilworth number. *Order*, 20(4):351–364, 2003.

[82] H. Fernau. Parameterized algorithmics: A graph-theoretic approach, 2005.

[83] J. Fiala, P. A. Golovach, and J. Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theoretical Computer Science*, 412(23):2513–2523, 2011.

[84] C. A. Floudas and X. Lin. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Annals of Operations Research*, 139(1):131–162, 2005.

[85] J. Flum and M. Grohe. The parameterized complexity of counting problems. *SIAM J. Computing*, 33(4):892–922, 2004.

[86] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[87] F. V. Fomin, B. M. P. Jansen, and M. Pilipczuk. Preprocessing subgraph and minor problems: When does a small vertex cover help? *J. Computer and System Sciences*, 80(2):468–495, 2014.

[88] F. V. Fomin, D. Lokshtanov, N. Misra, and S. Saurabh. Planar F-deletion: Approximation, kernelization and optimal FPT algorithms. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 470–479. IEEE Computer Society, 2012.

[89] F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Bidimensionality and kernels. In M. Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 503–510. SIAM, 2010.

[90] A. Frank and É. Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.

[91] C. P. Gabor, K. J. Supowit, and W. Hsu. Recognizing circle graphs in polynomial time. *J. ACM*, 36(3):435–473, 1989.

[92] J. Gajarský, P. Hlinený, J. Obdrzálek, S. Ordyniak, F. Reidl, P. Rossmanith, F. S. Villaamil, and S. Sikdar. Kernelization using structural parameters on sparse graph classes. *J. Computer and System Sciences*, 84:219–242, 2017.

[93] R. Ganian and P. Hlinený. On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. *Discrete Applied Mathematics*, 158(7):851–867, 2010.

[94] R. Ganian, P. Hlinený, and J. Obdrzálek. Better algorithms for satisfiability problems for formulas of bounded rank-width. *Fundamenta Informaticae*, 123(1):59–76, 2013.

[95] R. Ganian and S. Ordyniak. The complexity landscape of decompositional parameters for ILP. In D. Schuurmans and M. P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 710–716. AAAI Press, 2016.

[96] R. Ganian, S. Ordyniak, and M. S. Ramanujan. Going beyond primal treewidth for (M)ILP. In S. P. Singh and S. Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 815–821. AAAI Press, 2017.

[97] R. Ganian, F. Slivovsky, and S. Szeider. Meta-kernelization with structural parameters. *J. Computer and System Sciences*, 82(2):333–346, 2016.

[98] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[99] S. Gaspers, N. Misra, S. Ordyniak, S. Szeider, and S. Zivny. Backdoors into heterogeneous classes of SAT and CSP. *J. Computer and System Sciences*, 85:38–56, 2017.

[100] S. Gaspers and S. Szeider. Backdoors to satisfaction. In H. L. Bodlaender, R. Downey, F. V. Fomin, and D. Marx, editors, *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 287–317. Springer, 2012.

[101] E. Gassner and J. Hatzl. A parity domination problem in graphs with bounded treewidth and distance-hereditary graphs. *Computing*, 82(2-3):171–187, 2008.

[102] C. Gavoille and C. Paul. Distance labeling scheme and split decomposition. *Discrete Mathematics*, 273(1-3):115–130, 2003.

[103] M. U. Gerber and V. V. Lozin. Robust algorithms for the stable set problem. *Graphs and Combinatorics*, 19(3):347–356, 2003.

[104] E. Gioan and C. Paul. Dynamic distance hereditary graphs using split decomposition. In *Algorithms and Computation*, volume 4835 of *LNCS*, pages 41–51. Springer Verlag, 2007.

[105] E. Gioan and C. Paul. Split decomposition and graph-labelled trees: Characterizations and fully dynamic algorithms for totally decomposable graphs. *Discrete Applied Mathematics*, 160(6):708–733, 2012.

[106] E. Gioan, C. Paul, M. Tedder, and D. G. Corneil. Practical and efficient split decomposition via graph-labelled trees. *Algorithmica*, 69(4):789–843, 2014.

[107] P. A. Golovach, D. Paulusma, and J. Song. Closing complexity gaps for coloring problems on H-free graphs. *Information and Computation*, 237:204–214, 2014.

[108] E. Grädel, P. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Y. Vardi, Y. Venema, and S. Weinstein. *Finite Model Theory and Its Applications.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2007.

[109] J. L. Gross and T. W. Tucker. *Topological Graph Theory.* Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, 1987.

[110] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007.

[111] F. Gurski. The behavior of clique-width under graph operations and graph transformations. *Theory of Computing Systems*, 60(2):346–376, 2017.

[112] G. Gutin, M. Jones, and M. Wahlström. Structural parameterizations of the mixed chinese postman problem. In N. Bansal and I. Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 668–679. Springer, 2015.

[113] M. Habib and R. H. Möhring. On some complexity properties of N-free posets and posets with bounded decomposition diameter. *Discrete Mathematics*, 63(2-3):157–182, 1987.

[114] M. Habib and R. H. Möhring. Treewidth of cocomparability graphs and a new order-theoretic parameter. *Order*, 11(1):47–60, 1994.

[115] P. Heggernes, P. van 't Hof, B. M. P. Jansen, S. Kratsch, and Y. Villanger. Parameterized complexity of vertex deletion into perfect graph classes. *Theoretical Computer Science*, 511:172–180, 2013.

[116] R. Hemmecke, M. Köppe, and R. Weismantel. A polynomial-time algorithm for optimizing over N-fold 4-block decomposable integer programs. In F. Eisenbrand and F. B. Shepherd, editors, *Integer Programming and Combinatorial Optimization, 14th International Conference, IPCO 2010, Lausanne, Switzerland, June 9-11, 2010. Proceedings*, volume 6080 of *Lecture Notes in Computer Science*, pages 219–229. Springer, 2010.

[117] R. Hemmecke, S. Onn, and L. Romanchuk. n-Fold integer programming in cubic time. *Mathematical Programming*, 137(1-2):325–341, 2013.

[118] P. Hlinený and S. Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Computing*, 38(3):1012–1032, 2008.

[119] E. Howorka. A characterization of distance-hereditary graphs. *The Quarterly J. Mathematics*, 28(4):417–420, 1977.

[120] D. Hrabec, P. Popela, J. Roupec, J. Mazal, and P. Stodola. *Two-Stage Stochastic Programming for Transportation Network Design Problem*, page 17–25. Springer International Publishing, Cham, 2015.

[121] S. Hsieh, C. Ho, T. Hsu, and M. Ko. The Hamiltonian problem on distance-hereditary graphs. *Discrete Applied Mathematics*, 154(3):508–524, 2006.

[122] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory of Computing Systems*, 47(1):196–217, 2010.

[123] R. Hung and M. Chang. Linear-time algorithms for the Hamiltonian problems on distance-hereditary graphs' . *Theoretical Computer Science*, 341(1-3):411–440, 2005.

[124] R. Hung and M. Chang. Finding a minimum path cover of a distance-hereditary graph in polynomial time. *Discrete Applied Mathematics*, 155(17):2242–2256, 2007.

[125] T. Husfeldt, R. Paturi, G. B. Sorkin, and R. Williams. Exponential algorithms: Algorithms and complexity beyond polynomial time (dagstuhl seminar 13331). *Dagstuhl Reports*, 3(8):40–72, 2013.

[126] R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *J. Computer and System Sciences*, 62(2):367–375, 2001.

[127] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Computer and System Sciences*, 63(4):512–530, 2001.

[128] B. M. P. Jansen and H. L. Bodlaender. Vertex cover kernelization revisited - upper and lower bounds for a refined parameter. *Theoretical Computer Science*, 53(2):263–299, 2013.

[129] B. M. P. Jansen and S. Kratsch. A structural approach to kernels for ILPs: Treewidth and total unimodularity. In N. Bansal and I. Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 779–791. Springer, 2015.

[130] T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Directed tree-width. *J. Combinatorial Theory, Series B*, 82(1):138–154, 2001.

[131] K. Kangas, T. Hankala, T. M. Niinimäki, and M. Koivisto. Counting linear extensions of sparse posets. In S. Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 603–609. IJCAI/AAAI Press, 2016.

[132] R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.

[133] M. M. Kanté, E. J. Kim, O. Kwon, and C. Paul. An FPT algorithm and a polynomial kernel for linear rankwidth-1 vertex deletion. *Algorithmica*, 79(1):66–95, 2017.

[134] K. Kawarabayashi, Y. Kobayashi, and B. A. Reed. The disjoint paths problem in quadratic time. *J. Combinatorial Theory, Series B*, 102(2):424–435, 2012.

[135] K. Kawarabayashi and S. Kreutzer. The directed grid theorem. In R. A. Servedio and R. Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 655–664. ACM, 2015.

[136] E. J. Kim and O. Kwon. A polynomial kernel for block graph deletion. *Algorithmica*, 79(1):251–270, 2017.

[137] E. J. Kim and O. Kwon. A polynomial kernel for distance-hereditary vertex deletion. In F. Ellen, A. Kolokolova, and J. Sack, editors, *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, volume 10389 of *Lecture Notes in Computer Science*, pages 509–520. Springer, 2017.

[138] E. J. Kim, A. Langer, C. Paul, F. Reidl, P. Rossmanith, I. Sau, and S. Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Transactions on Algorithms*, 12(2):21:1–21:41, 2016.

[139] J. M. Kleinberg and É. Tardos. *Algorithm design.* Addison-Wesley, 2006.

[140] H. Kleine Büning and T. Lettmann. *Propositional Logic: Deduction and Algorithms.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1999.

[141] T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science.* Springer, 1994.

[142] D. Knop and M. Koutecký. Scheduling meets n-fold integer programming. *CoRR*, abs/1603.02611, 2016.

[143] D. Knop, M. Koutecký, and M. Mnich. Voting and bribing in single-exponential time. In H. Vollmer and B. Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 46:1–46:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[144] M. Kochol, V. V. Lozin, and B. Randerath. The 3-colorability problem on graphs with maximum degree four. *SIAM J. Computing*, 32(5):1128–1139, 2003.

[145] D. Korobitsyn. On the complexity of determining the domination number in monogenic classes of graphs. *Diskretnaya Matematika*, 2(3):90–96, 1990. In Russian, translation in Discrete Mathematics and Applications 2(2):191–199,1992.

[146] H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[147] L. A. Levin. Universal sequential search problems. *Probl. Peredachi Inf.*, 9:115–116, 1973.

[148] L. Libkin. *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[149] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002.

[150] D. Lokshantov, M. Vatshelle, and Y. Villanger. Independent set in P_5-free graphs in polynomial time. In C. Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 570–581. SIAM, 2014.

[151] F. Lonsing and A. Biere. Integrating dependency schemes in search-based QBF solvers. In O. Strichman and S. Szeider, editors, *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6175 of *Lecture Notes in Computer Science*, pages 158–171. Springer, 2010.

[152] K. D. Loof, H. D. Meyer, and B. D. Baets. Exploiting the lattice of ideals representation of a poset. *Fundamenta Informaticae*, 71(2-3):309–321, 2006.

[153] T. Lukasiewicz, M. V. Martinez, and G. I. Simari. Probabilistic preference logic networks. In T. Schaub, G. Friedrich, and B. O'Sullivan, editors, *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 561–566. IOS Press, 2014.

[154] H. Mannila and C. Meek. Global partial orders from sequential data. In R. Ramakrishnan, S. J. Stolfo, R. J. Bayardo, and I. Parsa, editors, *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000*, pages 161–168. ACM, 2000.

[155] G. J. Minty. On maximal independent sets of vertices in claw-free graphs. *J. Combinatorial Theory, Series B*, 28(3):284–304, 1980.

[156] R. H. Möhring. Computationally tractable classes of ordered sets. In I. Rival, editor, *Algorithms and Order*, volume 255 of *Nato Science Series C:*, pages 105–193. Springer Netherlands, 1989.

[157] J. Morton, L. Pachter, A. Shiu, B. Sturmfels, and O. Wienand. Convex rank tests and semigraphoids. *SIAM J. Discrete Mathematics*, 23(3):1117–1134, 2009.

[158] S. Nakano, R. Uehara, and T. Uno. A new approach to graph recognition and applications to distance-hereditary graphs. *J. Computer Science and Technology*, 24(3):517–533, 2009.

[159] J. Nesetril and P. O. de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European J. Combinatorics*, 27(6):1022–1041, 2006.

[160] J. Nesetril and P. O. de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.

[161] J. Nesetril and S. Shelah. On the order of countable graphs. *European J. Combinatorics*, 24(6):649–663, 2003.

[162] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford Lecture Series in Mathematics and Its Applications. OUP Oxford, 2006.

[163] T. M. Niinimäki and M. Koivisto. Annealed importance sampling for structure learning in bayesian networks. In F. Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 1579–1585. IJCAI/AAAI, 2013.

[164] Y. Okamoto, T. Uno, and R. Uehara. Counting the number of independent sets in chordal graphs. *J. Discrete Algorithms*, 6(2):229–242, 2008.

[165] S. Onn. *Nonlinear Discrete Optimization: An Algorithmic Theory.* Zurich lectures in advanced mathematics. European Mathematical Society Publishing House, 2010.

[166] C. Otwell, A. Remshagen, and K. Truemper. An effective QBF solver for planning problems. In H. R. Arabnia, R. Joshua, I. A. Ajwa, and G. A. Gravvanis, editors, *Proceedings of the International Conference on Modeling, Simulation & Visualization Methods, MSV '04 & Proceedings of the International Conference on Algorithmic Mathematics & Computer Science, AMCS '04, June 21-24, 2004, Las Vegas, Nevada, USA*, pages 311–316. CSREA Press, 2004.

[167] S. Oum. Rank-width and vertex-minors. *J. Combinatorial Theory, Series B*, 95(1):79–100, 2005.

[168] S. Oum and P. D. Seymour. Approximating clique-width and branch-width. *J. Combinatorial Theory, Series B*, 96(4):514–528, 2006.

[169] C. H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[170] D. Paulusma, F. Slivovsky, and S. Szeider. Model counting for CNF formulas of bounded modular treewidth. *Algorithmica*, 76(1):168–194, 2016.

[171] M. Peczarski. New results in minimum-comparison sorting. *Algorithmica*, 40(2):133–145, 2004.

[172] K. Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. Computer and System Sciences*, 67(4):757–771, 2003.

[173] W. B. Powell and H. Topaloglu. Stochastic programming in transportation and logistics. *Handbooks in Operations Research and Management Science*, 10:555–635, 2003.

[174] B. A. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

[175] J. Rintanen. Constructing conditional plans by a theorem-prover. *J. Artificial Intelligence Research*, 10:323–352, 1999.

[176] N. Robertson and P. D. Seymour. Graph minors. I. excluding a forest. *J. Combinatorial Theory, Series B*, 35(1):39–61, 1983.

[177] N. Robertson and P. D. Seymour. Graph minors. III. planar tree-width. *J. Combinatorial Theory, Series B*, 36(1):49–64, 1984.

[178] N. Robertson and P. D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

[179] N. Robertson and P. D. Seymour. Graph minors. X. obstructions to tree-decomposition. *J. Combinatorial Theory, Series B*, 52(2):153–190, 1991.

[180] F. Rosamond. Table of races, 2010. `http://fpt.wikidot.com/`.

[181] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Computing*, 5(2):266–283, 1976.

[182] A. Sabharwal, C. Ansótegui, C. P. Gomes, J. W. Hart, and B. Selman. QBF modeling: Exploiting player symmetry for simplicity and efficiency. In A. Biere and C. P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 382–395. Springer, 2006.

[183] M. Samer and S. Szeider. Backdoor sets of quantified boolean formulas. *J. Automated Reasoning*, 42(1):77–97, 2009.

[184] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Computer and System Sciences*, 4(2):177–192, 1970.

[185] A. Schrijver. *Theory of linear and integer programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, 1999.

[186] F. Slivovsky and S. Szeider. Variable dependencies and Q-resolution. In C. Sinz and U. Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 269–284. Springer, 2014.

[187] F. Slivovsky and S. Szeider. Quantifier reordering for QBF. *J. Automated Reasoning*, 56(4):459–477, 2016.

[188] D. A. Spielman and M. Bóna. An infinite antichain of permutations. *The Electronic J. Combinatorics*, 7, 2000.

[189] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In A. V. Aho, A. Borodin, R. L. Constable, R. W. Floyd, M. A. Harrison, R. M. Karp, and H. R. Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 1–9. ACM, 1973.

[190] S. Szeider. On fixed-parameter tractable parameterizations of SAT. In E. Giunchiglia and A. Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2003.

[191] H. Tamaki. A polynomial time algorithm for bounded directed pathwidth. In P. Kolman and J. Kratochvíl, editors, *Graph-Theoretic Concepts in Computer Science - 37th International Workshop, WG 2011, Teplá Monastery, Czech Republic, June 21-24, 2011. Revised Papers*, volume 6986 of *Lecture Notes in Computer Science*, pages 331–342. Springer, 2011.

[192] M. Tedder and D. G. Corneil. An optimal, edges-only fully dynamic algorithm for distance-hereditary graphs. In W. Thomas and P. Weil, editors, *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, volume 4393 of *Lecture Notes in Computer Science*, pages 344–355. Springer, 2007.

[193] S. Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Transactions on Algorithms*, 6(2), 2010.

[194] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem.* Discrete Mathematics and Applications. SIAM, Philadelphia, PA, USA, 2002.

[195] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

[196] M. van den Briel, T. Vossen, and S. Kambhampati. Reviving integer programming approaches for AI planning: A branch-and-cut framework. In S. Biundo, K. L. Myers, and K. Rajan, editors, *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA*, pages 310–319. AAAI, 2005.

[197] T. Vossen, M. O. Ball, A. Lotem, and D. S. Nau. On the use of integer programming models in AI planning. In T. Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 304–309. Morgan Kaufmann, 1999.