

Bounded Width Graph Classes in Parameterized Algorithms

Lars Jaffke



Dissertation for the degree of philosophiae doctor (PhD)
at the University of Bergen, Norway

2020

Defended: August 26, 2020

Acknowledgements

I have been fortunate to have two advisors with very different philosophies; I am deeply grateful to each of them. Mike Fellows, for always passionately encouraging me to pursue my own ideas, with insight and broad perspective. Thank you for continuously and generously sharing your new ideas, and for always having time for discussions. Jan Arne Telle, for your close guidance in the early stages of my PhD, the many hours spent together on the whiteboard, and for letting me follow my own path when you saw that I was ready. Thank you for your open door, always taking time for me, and for advice that often went far beyond technical, mathematical issues. I would also like to thank Frances Rosamond at this point, who with relentless drive and energy has been an ever-present source of inspiration and support.

I thank the Trond Mohn Foundation for providing the funding for my position, and Mike for the generous travel allowance, through which I could attend many conferences and other events that helped me grow as a researcher. I would like to thank the evaluation committee of this thesis; Meirav Zehavi and Dimitrios Thilikos for assessing my work, and Ahmad Hemmati for being the committee leader.

Mathematics is a team sport, and none of this work would have been possible without the many fantastic colleagues I got to collaborate with in the last years, in addition to my advisors. First I would like to name O-joung Kwon, who besides being a brilliant researcher is also a true pleasure to work with. Thank you for the fruitful collaboration, and for the two invitations to Incheon. I am grateful to Mateus de Oliveira Oliveira, for always being keen one venturing into unexplored areas and exciting corners of our research field, and many fun discussions and projects. Thank you Geevarghese Philip, both for being a driving force in bringing ‘diversity’ to the FPT-world, and for the invitation to Chennai in Spring this year. Paloma Lima, for making my work much more colorful — of course not only my work, but more about that later. I am sincerely grateful to Hans Bodlaender, for what I find the most beautiful result in this thesis – the Merge Dominator Lemma – and for the associated visit in my former home, Utrecht, which I enjoyed very much. Last but not least I thank my co-authors Jungho Ahn, Julien Baste, Tesshu Hanaka, Pinar Heggernes, Bart M. P. Jansen, Aliz Király, Daniel Lokshtanov, Tomáš Masařík, Hirotaka Ono, Yota Otachi, Frances Rosamond, Günter Rote, Torstein J. F. Strømme, Hans Raj Tiwary, Mathias Weller, and Tom van der Zanden.

Quite possibly I would not have developed such a deep interest in algorithms if it was not for a bold move of three researchers in information systems to make use of complicated combinatorial algorithms in the context of business process model analysis. I thank Patrick Delfmann, Hanns-Alexander Dietrich, and Matthias Steinhost for introducing me to the notions of treewidth and fixed-parameter tractability, which are still at the core of my research interests. Treewidth inspired my move to Utrecht where I studied with Hans Bodlaender, whom I feel privileged to call a mentor until the present day.

The algorithms group at UiB provided a motivating and stimulating atmosphere, and I consider myself lucky to have been part of it. Thanks to all its professors and students, and to all fellow PhD students and postdocs I have had the pleasure to meet in the last years. I am especially grateful to everybody I met there or at conferences and other events with whom I could share a walk/a drink/a laugh: Tom v. d. Z., Astrid, Sándor, Tom B., Srinivasan, Amer, Carl, Eduard, Tomáš, Pratibha, Roohani, Benjamin, and funnest office mate Kristine. I very much value the experiences I made as a teaching assistant for several courses, and would like to thank Anya Helene Bagge, Fedor Fomin, and Pinar Heggernes for having me as a TA; and of course the students for participating in the group sessions. I also want to thank the administration staff at UiB that has always been extremely helpful and efficient.

An advantage of academic life is that you have friends all over the world; a disadvantage of academic life is that your friends are spread all over the world. I am extremely thankful for every friend I manage to keep in touch with, to whichever extent. Too many come to mind, but I would like to name a few. First and foremost thank you Charis, for your friendship, and for a truly special [agent 007 James...]. For the countless batches of co-cooked Bolognese, and all the other good memories. Thank you Krystallia, Popi, Marina, Christian, Lubo, Alessandro, Peter, Miriam, Freddi, Guido, Carsten, and Mose. And my friends from back home, Volker, Christoph, Martin, Flo, the *Gang* in Malsch; Philipp and of course my oldest friend Mobbet. I am also truly grateful for the friends I have made in Bergen. Sindre, for nerding together about music and beer and many nights out, together with Maria. Trym, for fun and weird nights, and all the afternoons/evenings at Café Opera.

Paloma. I cannot express how grateful I am that you came into my life. Thank you for your love, for being my companion, and for always being there for me; for every beautiful day in our little home by the sea.

My most heartfelt gratitude goes to my family whose support and comfort has been invaluable to me my whole life. To my grandparents, my aunts and uncles, my cousins, to my brother, my mother and my father. Thank you Björn for everything I have learned from you; for not only being a brother, but also a friend. Thank you Gaby and Thomas, for your unconditional love and support, your countless efforts towards me, for always believing in me and encouraging me, especially in times when I myself could not see my way. None of this would have been possible otherwise, so I am happy to dedicate this work to you.

Abstract

A common way to tackle the computational intractability of combinatorial problems on graphs is to impose restrictions on the input graphs that can be exploited in the design of efficient algorithms. Two mainstream ways to impose such restrictions are (*i*) to require the input graph to come from a certain *graph class*, or (*ii*) to assume that the input graph has *small width* according to some width measure. While these two approaches are typically considered separately in the literature, it has been observed that width measures with high expressive power are bounded by a constant on several well-studied graph classes. In these cases, efficient algorithms for graphs of small width unify and extend tractability results on graph classes.

The main body of this thesis is divided into three parts. In the first part, we briefly discuss the width measures *treewidth*, *clique-width*, and *maximum induced matching width*, mim-width for short, and survey some graph classes whose width is bounded (or unbounded) in terms of one of these measures. In the second part, we prove a structural lemma on integer sequences that addresses a runtime bottleneck of the currently fastest FPT-algorithms to compute treewidth and pathwidth. However, on its own, it does not lead to an asymptotic improvement in the runtime. We apply this lemma to give polynomial-time algorithms to compute the directed cutwidth and directed modified cutwidth of series-parallel digraphs. In the third part, we give XP-time algorithms for *non-local* problems parameterized by mim-width, meaning problems whose solutions cannot be verified by inspecting the direct neighborhood of all vertices independently. We consider problems concerned with finding induced paths in graphs as well as the FEEDBACK VERTEX SET problem. We then show that several problems related to domination and independence are W[1]-hard parameterized by mim-width. Finally, we study variants of vertex-coloring problems, namely *b*-COLORING and CLIQUE COLORING. We give XP-algorithms for both problems parameterized by clique-width, and study the fine-grained complexity of CLIQUE COLORING parameterized by treewidth.

Contents

Acknowledgements	i
Abstract	iii
1 Introduction	1
I Basic Concepts	7
2 Fundamentals	9
2.1 Your Graph is My Graph	9
2.2 Parameterized Complexity	10
2.3 CNF-SAT and Exponential Time Hypotheses	13
3 Width Measures	15
3.1 Treewidth and Pathwidth	15
3.1.1 Computing Treewidth	20
3.2 Branch Decompositions	22
3.3 Clique-Width and Module-Width	26
3.3.1 Comparison with Treewidth	34
3.3.2 Computing Clique-Width	34
3.4 Mim-Width	36
3.4.1 Efficiently Solvable Problems	43
3.4.2 Comparison with Clique-Width	45
3.4.3 Computing Mim-Width	45
3.5 Monadic Second Order Logic of Graphs	46
3.6 Generalized (Distance) Domination Problems	51
3.6.1 (σ, ρ) -Domination	52
3.6.2 Distance- r (σ, ρ) -Domination	55
3.6.3 Locally Checkable Vertex Partitioning Problems	56

4 Graph Classes	59
4.1 Subclasses of Planar Graphs	59
4.2 Chordal Graphs	61
4.3 Intersection Graphs	64
4.4 Recursively Generated Graphs (and Relatives)	70
4.5 Graphs Related to Orderings	72
4.6 Forbidden Induced Subgraph Characterizations	73
4.7 Width Bounds	76
4.7.1 Treewidth Bounds	77
4.7.2 Clique-Width Bounds	77
4.7.3 Mim-Width Bounds	80
4.7.4 Sim-Width	85
4.7.5 Bounds for \mathcal{H} -Free Graphs	86
5 Mim-Width of Graph Powers	89
5.1 General Graphs	90
5.2 Graphs of Bounded Treewidth or Clique-Width	92
5.3 Leaf Power Graphs	95
II Computing Width Measures	97
6 Typical Sequences Revisited	99
6.1 Typical Sequences	100
6.2 The Merge Dominator Lemma	107
6.2.1 The Merge Matrix, Paths, and Non-Diagonality	108
6.2.2 The Split Lemma	111
6.2.3 The Chop Lemmas	113
6.2.4 The Split-and-Chop Algorithm	120
6.2.5 Generalization to Arbitrary Integer Sequences	121
6.3 Employing the Merge Dominator Lemma	123
7 Width Measures of Series Parallel Digraphs	125
7.1 Cutwidth of SPDs	127
7.2 Modified Cutwidth of SPDs	132
7.3 Further Applications of the MDL?	135
III Computing With Width Measures	137
8 Non-Local Problems on Mim-Width	139
8.1 Notation and Minimal Vertex Covers	139
8.2 Induced Path Problems on Mim-Width	141

8.2.1	Longest Induced Path	142
8.2.2	Induced Disjoint Paths	152
8.2.3	H -Induced Topological Minor	157
8.3	Feedback Vertex Set on Mim-Width	159
8.3.1	Reduced Forests and Vertex Covers	161
8.3.2	The Algorithm	170
8.4	Some Concluding Remarks	177
9	Hardness Results on Mim-Width	179
9.1	Hamiltonian Cycle on Linear Mim-Width 1	179
9.2	Hardness Results for H -Graphs due to Fomin et al.	181
9.2.1	Independent Set	182
9.2.2	Dominating Set	184
9.3	$W[1]$ -Hard (σ, ρ) -Domination Problems by Mim-Width	185
9.3.1	Maximization Problems	185
9.3.2	Minimization Problems	191
9.4	FVS is $W[1]$ -Hard by Mim-Width	196
9.5	Discussion and Open Problems	201
10	Coloring Problems on Clique-Width	203
10.1	b -Coloring	205
10.1.1	Outline of the Algorithm	208
10.1.2	t -Types and t -Signatures	210
10.1.3	Compatibility	212
10.1.4	Merging and Splitting Partial b -Colorings	215
10.1.5	The Algorithm	220
10.1.6	b -Continuity	222
10.2	Fall Coloring	223
10.3	Clique Coloring	225
10.3.1	Outline of the Algorithm	226
10.3.2	Potentially Bad Cliques	227
10.3.3	The Type of a Color Class	231
10.3.4	The Algorithm	233
10.4	Conclusion and Open Problems	237
11	Fine-Grained Complexity of Clique Coloring on Treewidth	241
11.1	Upper Bound	242
11.2	Lower Bound	246
11.2.1	The Case $q = 2$	247
11.2.2	q -LIST COLORING Parameterized by Treewidth	249
11.2.3	The Case $q \geq 3$	254

IV	Further Research	257
V	Appendix	287
A	Dictionary	289
A.1	Graphs	291
A.2	Digraphs	294
A.3	Asymptotics, Runtime, and Complexity	294

Introduction

Graphs are a great tool to model binary relations and in one way or another, we encounter them almost every day. Transportation networks and information networks can be modeled as graphs. Social networks can be modeled as graphs. Graphs are useful in the analysis of language, the structure of molecules, nervous systems, protein interactions, maps. Graphs are of help in the design of electrical circuits, in planning and scheduling, and in the visualization of relational data. These are just a few examples baring witness that since its humble beginnings in the 18th century [115], the study of graphs has become a central discipline in many areas.

Let us get a bit more concrete. Suppose we have a set of objects, in which some pairs are in *conflict* with each other. Think for instance about a set of tasks, and a conflict meaning two tasks *cannot be performed at the same time*, so that we can perform at most one of the two. Or a set of people, and a conflict simply meaning two people *do not get along*. In both scenarios, we may want to find a *largest conflict-free subset* of the objects. In the first case this corresponds to a largest subset of tasks that we can possibly perform, and in the second case to a largest group of people we can gather in one place without risking a fight. A natural model for both scenarios is to create a graph that contains one vertex for each element, and an edge between two vertices whenever the corresponding elements are in conflict. Finding a largest conflict-free subset then corresponds to finding a *maximum independent set*, i.e. a set of pairwise non-adjacent vertices of maximum size, in the resulting graph. Unfortunately, this computational problem known as MAXIMUM INDEPENDENT SET turns out to be computationally intractable (NP-hard) [179], meaning that we cannot expect to have an efficient (polynomial-time) algorithm that solves all instances of the problem optimally.

This is the case not only for the MAXIMUM INDEPENDENT SET problem, but for most interesting computational problems on graphs. One way to cope with this intractability barrier is to relax the requirement that *every* instance of the problem has to be solvable efficiently to only a *certain subset* of all instances. In this thesis,

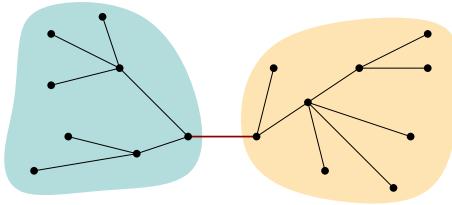


Figure 1.1: A tree being disconnected into two components by the removal of an edge.

we consider the two (arguably) most prominent approaches in this direction, namely the restrictions to *graph classes* and to *graphs of bounded width*, and the largely unexplored connections between the two.

A *graph class* is a set of graphs that share a common property. For instance, the class of *planar* graphs are the graphs that can be drawn in the plane without any pair of edges crossing. Graphs arising from applications may naturally fall into one such class. For instance, going back to the ‘conflict-freeness’ theme, suppose you are visiting a congress, with many events scheduled in some fixed time frame. To make the most of it, you want to attend as many events as possible, while clearly, you can only visit one event at any given time. Again we can create a *conflict graph* whose vertices are the events with an edge between two vertices whenever the corresponding events overlap in time. This graph belongs to the class of *interval* graphs, and it is well-known that the MAXIMUM INDEPENDENT SET problem is efficiently solvable in interval graphs, see for instance [258]. On the more general class of 2-*interval* graphs, the problem remains intractable [12].

In this vein, for each combination of a graph problem that is intractable in general and a graph class, we can determine that either the problem becomes efficiently solvable when restricted to the class, or that it remains hard. The body of literature on such results is vast; the reference database [252] currently lists 1610 classes and 218912 inclusions. This sheer volume makes the search for theories that unify algorithmic results on graph classes a worthwhile endeavour; and recent work has shown that *width measures* possess a power to provide such unification theorems that has widely gone unharnessed so far.

A width measure is typically defined via a numerical invariant (the width) of some kind of *decomposition* of graphs. The idea is that if a graph admits a decomposition of small width, then it is structurally simple. A prime example of structurally simple graphs is that of trees, see Figure 1.1. Removing any edge in a tree results in a graph with two connected components that are connected only via this edge. This property of trees can often be exploited to obtain efficient algorithms for problems that are intractable in general. The width measure *treewidth*, which may be regarded as the archetype of width measures of graphs, in some sense captures how close a graph is to being a tree. The corresponding *tree decomposition* shows us how to traverse the vertices of the graph along small (with respect to the width) subsets that disconnect

the graph, in analogy with the edges of a tree. Many polynomial-time algorithms for trees can be generalized to polynomial-time algorithms on graphs whose treewidth is bounded by some small constant.

A downside of treewidth is that it takes on large values in graphs with many edges, even if they have an extremely simple structure. For instance, the presence of a large *clique* (a subset of pairwise adjacent vertices) immediately implies large treewidth. The width measure *clique-width* was defined to overcome this issue. Early on, it was observed that the clique-width is bounded by a constant in several non-trivial graph classes that had already received much attention on their own [148]. Therefore, algorithms whose running times are polynomial on graphs with constant clique-width imply polynomial-time algorithms for such graph classes. On the negative side, (in the same work [148]) clique-width was also shown to remain unbounded on several well-structured graph classes; for instance in the above mentioned class of interval graphs.

The recent introduction of the measure *mim-width* [281] added a powerful tool for unification theorems of algorithmic results on graph classes to the (sparsely populated) toolbox. Many deeply studied graph classes, including interval graphs and several of their generalizations, were shown to have constant mim-width. With its introduction, a large number of problems related to independence and domination (including MAXIMUM INDEPENDENT SET) was shown to be efficiently solvable on graphs of small mim-width. This provided unified polynomial-time algorithms for many graph classes at once,¹ for a large number of computational problems.

The focus of this thesis is on efficient algorithms for hard problems on graphs of small width. In particular, we consider intractable graph problems *parameterized* by the width of the input graph. The philosophy of the field of parameterized complexity [104] is to consider a secondary measure of the input – the parameter, denoted here by k – and design algorithms whose exponential term of the runtime depends on the parameter alone. Such algorithms, called *fixed-parameter tractable* or FPT, run in time $f(k) \cdot n^c$, for some computable function f , input size n , and fixed constant c , and remain efficient for small values of k .

Other types of parameterized algorithms run in (XP) time $n^{f(k)}$, where again n is the input size and f some computable function. For fixed values of k , this is still polynomial time, but the distinction between FPT and XP running times is essential for both theoretical and practical purposes [105, 93]. Due to the high expressive power of the parameters clique-width and mim-width, the fastest running times we can obtain for problems parameterized by clique-width or mim-width are often XP. This still gives unifying polynomial-time results on graph classes of small width.

Outline of the Thesis and Contributions. This thesis is divided into four parts plus an appendix. In Part I, we introduce the width measures and graph classes that are of importance for this work, give short crash courses in designing algorithms based

¹Figure 4.9 on page 87 provides an overview.

on bounded-width decompositions, and review the currently known width bounds on the graph classes of interest. It contains some new results regarding the mim-width of graph powers and some of their algorithmic and structural consequences. These results are published in [JKST19].

To harness algorithmic results using width measures, we need to know how to compute a bounded-width decomposition of the input graph first. More often than not, this poses a greater challenge than designing algorithms for basic problems that use decompositions. In the case of treewidth, the currently fastest exact FPT-algorithm to compute an optimal (width- k) tree decomposition of an n -vertex graph takes time $2^{\mathcal{O}(k^3)} \cdot n$ [29], and finding an algorithm with running time $2^{o(k^3)} \cdot n^{\mathcal{O}(1)}$ has been a central open problem in the field for more than 20 years. In Part II, published in [BJT20], we prove a lemma that addresses a runtime bottleneck in the algorithm of [29]. However, on its own it is not sufficient to give an asymptotic improvement. On the other hand, we show that we can use this lemma to compute width measures of series-parallel digraphs, a subclass of directed acyclic graphs.

Part III contains algorithms and lower bounds for several problems parameterized by the width of given decompositions. Chapter 8, whose results are published in [JKT20b, JKT20a], gives XP-algorithms for ‘non-local’ problems² parameterized by the mim-width of a given decomposition. In particular, the algorithms are for problems concerning induced paths in graphs, and FEEDBACK VERTEX SET. All algorithms known so far using the parameter mim-width run in XP time. Due to the high expressive power of this parameter, one may find FPT-algorithms parameterized by mim-width unlikely, but until recently no parameterized hardness results were known. Extending the work [127] in which the first hardness results parameterized by mim-width appeared, we rule out FPT-algorithms (under the standard assumption $\text{FPT} \neq \text{W}[1]$) for several problems, including FEEDBACK VERTEX SET, in Chapter 9. These proofs appear in the publications [JKT20a, JKST19].

In Chapter 10, we give XP-algorithms for vertex coloring problems parameterized by clique-width. Concretely, we give such algorithms for the b -COLORING (based on the preprint [JLL20+]) and CLIQUE COLORING (based on [JLP20]) problems. In the case of b -COLORING, this resolves two open questions regarding the complexity of the problem on graph classes stated independently in the literature [46, 69]. In Chapter 11 we study the *fine-grained* complexity of the CLIQUE COLORING problem on graphs of bounded treewidth. This chapter is based on excerpts from [JLP20] and [JJ17].

We conclude this thesis in Part IV. Below we list the publications to which the author of this thesis contributed. This work is largely based on the publications [JLP20, BJT20, JKT20b, JKT20a, JKST19], and the preprint [JLL20+].

²Here, by *local* we mean problems in which a potential solution can be verified by inspecting its interaction with the closed neighborhood of each vertex in the graph independently.

List of Publications

- [JLL20+] Lars Jaffke, Paloma T. Lima, and Daniel Lokshtanov. *b*-coloring parameterized by clique-width. preprint, arXiv:2003.04254, 2020+.
- [JLP20] Lars Jaffke, Paloma T. Lima, and Geevarghese Philip. Structural parameterizations of clique coloring. In Javier Esparza and Daniel Král', editors, *Proceedings of the 45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, 2020. To appear.
- [JOT20] Lars Jaffke, Mateus de Oliveira Oliveira, and Hans Raj Tiwary. Compressing permutation groups into grammars and polytopes. A graph embedding approach. In Javier Esparza and Daniel Král', editors, *Proceedings of the 45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, 2020. To appear.
- [AJKL20] Jungho Ahn, Lars Jaffke, O-joung Kwon, and Paloma T. Lima. Well-partitioned chordal graphs: Obstruction set and disjoint paths. In Isolde Adler and Haiko Müller, editors, *Proceedings of the 46th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2020)*, 2020. To appear.
- [BFJ⁺20] Julien Baste, Michael R. Fellows, Lars Jaffke, Tomáš Masařík, Mateus de Oliveira Oliveira, Geevarghese Philip, and Frances A. Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. In Christian Bessiere, editor, *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020)*, pages 1119–1125, 2020.
- [BHQJ⁺20] Hans L. Bodlaender, Tesshu Hanaka, Lars Jaffke, Hirotaka Ono, Yota Otachi, and Tom C. van der Zanden. Hedonic seat arrangement problems. In Bo An, Neil Yorke-Smith, Amal El Fallah Seghrouchni, and Gita Sukthankar, editors, *Proceedings of the 19th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2020)*, pages 1777–1779, 2020.
- [BJT20] Hans L. Bodlaender, Lars Jaffke, and Jan Arne Telle. Typical sequences revisited - Computing width parameters of graphs. In Christophe Paul and Markus Bläser, editors, *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, volume 154 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 57:1–57:16. Schloss Dagstuhl, 2020.
- [JL20] Lars Jaffke and Paloma T. Lima. A complexity dichotomy for critical values of the b-chromatic number of graphs. *Theoretical Computer Science*, 815:182–196, May 2020. Extended abstract at MFCS 2019.

- [JKT20b] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width I. Induced path problems. *Discrete Applied Mathematics*, 278:153–168, May 2020. Extended abstract at IPEC 2017.
- [JKT20a] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width II. The feedback vertex set problem. *Algorithmica*, 82:118–145, January 2020. Extended abstract at STACS 2018.
- [JKST19] Lars Jaffke, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. Mim-width III. Graph powers and generalized distance domination problems. *Theoretical Computer Science*, 796:216–236, December 2019. Extended abstract at IPEC 2018.
- [BJM⁺19] Julien Baste, Lars Jaffke, Tomáš Masařík, Geevarghese Philip, and Günter Rote. FPT algorithms for diverse collections of hitting sets. *Algorithms*, 12:254, November 2019.
- [FJK⁺18] Michael R. Fellows, Lars Jaffke, Aliz Király, Frances A. Rosamond, and Matthias Weller. What is known about vertex cover kernelization? In Hans-Joachim Böckenhauer, Dennis Komm, and Walter Unger, editors, *Adventures Between Lower Bounds and Higher Altitudes. Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, volume 11011 of *Lecture Notes in Computer Science (LNCS)*, pages 330–356. Springer, 2018.
- [JO18] Lars Jaffke and Mateus de Oliveira Oliveira. On weak isomorphism of rooted vertex-colored graphs. In Andreas Brandstädt, Ekkehard Köhler, and Klaus Meer, editors, *Proceedings of the 44th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2018)*, volume 11159 of *Lecture Notes in Computer Science (LNCS)*, pages 266–278. Springer, 2018.
- [JBHT17] Lars Jaffke, Hans L. Bodlaender, Pinar Heggernes, and Jan Arne Telle. Recognizability equals definability for k -outerplanar graphs and ℓ -chordal partial k -trees. *European Journal of Combinatorics*, 66:191–234, December 2017. Extended abstract at IPEC 2015.
- [JJ17] Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. In Dimitris Fotakis, Aris Pagourtzis, and Evangelis Th. Paschos, editors, *Proceedings of the 10th International Conference on Algorithms and Complexity, (CIAC 2017)*, volume 10236 of *Lecture Notes in Computer Science (LNCS)*, pages 345–356. Springer, 2017.

Part I

Basic Concepts

2

Fundamentals

In this chapter we fix some basic notation and terminology, and briefly introduce the most fundamental concepts in parameterized complexity, as well as the Exponential Time Hypothesis and the Strong Exponential Time Hypothesis. A catalogue of definitions is given in Appendix A.

Miscellaneous Notation. For a set Ω , we denote by $|\Omega|$ the size of Ω , i.e. the number of elements contained in Ω , and by 2^Ω the *power set*, i.e. the set of all subsets, of Ω . For an integer $k \leq |\Omega|$, we denote by $\binom{\Omega}{k}$ the set of all size- k subsets of Ω , and by $\binom{\Omega}{\leq k}$ the set of all subsets of Ω of size at most k .

For an equivalence relation \sim over a set Ω , and for each $x \in \Omega$, we denote by $[x]_\sim$ the equivalence class of x , i.e. the set $\{y \in \Omega \mid x \sim y\}$. We furthermore denote by Ω/\sim the set of all equivalence classes of \sim .

For integers a, b with $a < b$, we let $[a..b] := \{a, a + 1, \dots, b\}$ and for a positive integer a , we let $[a] := [1..a]$.

An *alphabet* Σ is a set of symbols, and we denote by Σ^* all strings whose characters are from Σ . For a string $x \in \Sigma^*$, we denote by $|x|$ the length of x .

The \mathcal{O}^* -notation suppresses polynomial factors in a variable that is always clear from the context.

2.1 Your Graph is My Graph

To make sure that reader and author are on the same page, we establish the notation for basic graph-theoretic concepts used throughout this thesis. We list their definitions in Appendix A.1, and follow standard conventions [45, 101].

Unless stated otherwise, a graph G with vertex set $V(G)$ and edge set $E(G)$ is finite, undirected, simple, and loop-free. We use the shorthand ‘ uv ’ to denote an edge $\{u, v\} \in E(G)$. We denote the endpoints of a set of edges $F \subseteq E(G)$ by

$V(F)$. For a vertex set X , $G[X]$ is the subgraph of G induced by X . We say that a graph G contains H (as an induced subgraph), if G has an induced subgraph that is isomorphic to H . For two disjoint vertex sets $X, Y \subseteq V(G)$, $G[X, Y]$ is the bipartite subgraph of G induced by (X, Y) .

For a vertex $v \in V(G)$, we denote by $N_G(v)$ its open neighborhood, by $N_G[v] := N_G(v) \cup \{v\}$ its closed neighborhood, and by $\deg_G(v)$ its degree. We denote by $\Delta(G)$ the maximum degree of G . A graph G is *subcubic* if $\Delta(G) \leq 3$. For a set $X \subseteq V(G)$, we let $N_G(X) := N_G(X) \setminus X$ and $N_G[X] := N_G(X) \cup X$. Unless stated otherwise,¹ the length of a path is the number of its edges and for two vertices $u, v \in V(G)$ in a graph G , we denote by $\text{dist}_G(u, v)$ the distance between u and v . We may drop G as a subscript in all cases if it clear from the context.

A *cut* of a graph G is a 2-partition of its vertex set. (Not to be confused with a *cut vertex* which is a vertex whose removal increases the number of connected components.)

A collection of pairwise disjoint edges $M \subseteq E(G)$ is called a *matching*, and M is called an *induced matching* if G contains no additional edges between any pair in of vertices from $V(M)$.

For a tree T , we denote by $L(T)$ the set of its leaves. If T is rooted, then for each vertex $v \in V(T)$, we denote by T_v the subtree of T rooted at v .

For $n, m \in \mathbb{N}$, we denote by K_n a complete graph on n vertices, by C_n a cycle on n vertices, by P_n a path on n vertices, by $K_{n,m}$ a complete bipartite graph whose vertex bipartition has parts of size n and m .

2.2 Parameterized Complexity

In many NP-hard combinatorial problems, the input consists of some structure X over a universe U and an integer k , and the question is whether there is a size- k subset of U that has some property with respect to X . Think for instance of the VERTEX COVER problem, where we are given a graph G over some size- n vertex set $V(G)$ and an integer k , and the question is whether there is some size- k set of vertices S such that each edge of G has at least one endpoint in S .

A naive approach to solve this problem is to enumerate all size- k vertex subsets of the graph, and test for each of them whether they form a vertex cover or not. This gives an algorithm that solves VERTEX COVER in time $\mathcal{O}(n^k \cdot m)$. If we consider k to be a constant, this is a polynomial running time. Nevertheless, since the degree of the polynomial depends on k , such a runtime is infeasible in practice already for relatively moderate values of k (say, $k \geq 10$).

Consider the following alternative idea to solve VERTEX COVER. When building a vertex cover, we know that for each vertex $v \in V(G)$, either v has to be inside the

¹At some point in Chapter 8 it is temporarily more convenient to use the number of vertices as the length of a path.

vertex cover, or all of its neighbors have to be in the vertex cover. (Otherwise, there is an edge that is not covered.) Therefore, to solve the given instance (G, k) , we pick up a vertex $v \in V(G)$, and recursively solve the instances $(G - v, k - 1)$ (for when we add v to the vertex cover) and $(G - N[v], k - \deg(v))$ (for when we add $N(v)$ but not v to the vertex cover). If we can guarantee at each step in the recursion that the graph of our instance has a vertex of degree at least 1, we know that the size-budget decreases by at least one in each recursive call. But this is immediate: a vertex u of degree 0 is not incident to any edge, and therefore irrelevant when constructing a vertex cover of the graph; we can simply remove u .

Now, if at some point in the recursion we have that $k \geq 0$ and the graph is reduced to the empty graph, then we know that the original instance had a vertex cover of size (at most) k : following the choices made in the recursive steps gives a vertex cover of size at most k . If on the other hand, $k = 0$, and the graph of the instance still has edges, then we can stop: the choices of vertices that lead to this point in the computation will not produce a vertex cover of size k , since it leaves us with some uncovered edges, and our budget is already used up.

The runtime of the algorithm is described by the recurrence $T(k) \leq 2T(k - 1) + \mathcal{O}(m)$, which with the base case of $T(0) = \mathcal{O}(1)$ resolves to $\mathcal{O}(2^k \cdot m)$. We obtained an algorithm where the degree of the polynomial *does not depend on k* , therefore the algorithm remains practical for much larger values of k , compared to the one we have seen above.

The distinction between problems that admit algorithms with such runtimes and the ones that do not is one of the main subjects in the field of *parameterized complexity*, whose systematic study was initiated by Downey and Fellows [104]. Concretely, in this field we consider *parameterized problems*, which are obtained from any computational problem Π by identifying a numerical measure of the inputs of Π , called the *parameter*. One of the main goals is to decide if a parameterized problem with parameter k admits an algorithm whose running time is of the form $f(k) \cdot n^c$, where f is some computable function, n the input size, and $c \in \mathbb{N}$ a fixed constant. Such a running time is called *fixed-parameter tractable*, and the class of parameterized problems admitting such algorithms is denoted by **FPT**.

A parameterized problem with parameter k is said to be in the complexity class **XP**, if it can be solved in time $n^{f(k)}$, where n is the input size and f some computable function. Clearly, **FPT** \subseteq **XP**. In the example of VERTEX COVER given above, the first algorithm with runtime $\mathcal{O}(n^k \cdot m)$ is an **XP**-algorithm, while the second algorithm with runtime $\mathcal{O}(2^k \cdot m)$ is an **FPT**-algorithm. Therefore, we have shown that VERTEX COVER parameterized by the solution size is in **FPT**. We formally summarize these notions in the following definition.

Definition 2.1 (FPT, XP). Let Σ be an alphabet. A *parameterized problem* is a set $\Pi \subseteq \Sigma^* \times \mathbb{N}$, the second component being the *parameter* which usually expresses a structural measure of the input.

FPT A parameterized problem Π is called *fixed-parameter tractable*, or said to be in

the complexity class **FPT**, if there is a computable function f and a constant c , such that there is an algorithm that for each $(x, k) \in \Sigma^* \times \mathbb{N}$, correctly decides whether or not $(x, k) \in \Pi$ in time $f(k) \cdot |x|^c$.

XP A parameterized problem Π is called *slice-wise polynomial*, or said to be in the complexity class **XP**, if there is a computable function f , such that there is an algorithm that for each $(x, k) \in \Sigma^* \times \mathbb{N}$, correctly decides whether or not $(x, k) \in \Pi$ in time $|x|^{f(k)}$.

While the solution size is a natural target for a parameter, we would like to stress that the notion of a parameterized problem is much more versatile. In other words, ‘anything’ can be a parameter. For instance width measures of graphs and other combinatorial objects, approximation ratios, numbers of variables for instance in an ILP context, alphabet sizes for problems on words, the number of dimensions in geometric or topological problems, etc etc – often even a *combination* of several such measures is meaningful.

The analogue of **NP**-hardness in parameterized complexity is that of hardness for the complexity class **W[1]** whose formal definition we omit, and instead refer to the textbooks [93, 104, 105]. The basic assumption is that **FPT** \neq **W[1]**, and if any **W[1]**-hard problem admits an **FPT**-algorithm, this would imply a violation of this assumption. For the sake of this thesis, it suffices to know that there are canonical **W[1]**-hard problems, and that hardness is transferred via parameterized reductions, which we discuss below. A canonical **W[1]**-hard problem is for instance the **CLIQUE** problem, which given a graph G and an integer k asks whether G contains a clique of size k , with k being the parameter.

Definition 2.2 (Parameterized Reduction). A *parameterized reduction* from a parameterized problem A to a parameterized problem Π is an algorithm that transforms each instance (x, k) of A to an instance (x', k') of Π such that

- (i) $(x, k) \in A$ if and only if $(x', k') \in \Pi$,
- (ii) $k' \leq g(k)$ for some computable function g , and
- (iii) the algorithm runs in time $f(k) \cdot |x|^c$ for some computable function f and constant c .

Now, if there is a parameterized reduction from A to Π with f , g , and c as in the previous definition, and Π has an **FPT**-algorithm running in time $h(k) \cdot |x|^d$, then A is **FPT** as well: Given an instance (x, k) of A , we use the reduction to obtain an instance (x', k') of Π , and we use the **FPT**-algorithm for Π on the instance (x', k') to decide whether or not $(x, k) \in A$. Correctness is immediate from the first item in the definition of a parameterized reduction. Observe that $|x'| \leq f(k) \cdot |x|^c$ and $k' \leq g(k)$.

Now, since²

$$h(k') \cdot |x'|^d \leq h(g(k)) \cdot (f(k) \cdot |x|^c)^d = h(g(k))f(k)^d \cdot |x|^{cd},$$

which is an FPT running time (take $f'(k) = h(g(k))f(k)^d$ which is a computable function), and the reduction takes FPT time, the whole process takes FPT time as well.

Theorem 2.3 ([93, 104, 105]). *If there is a parameterized reduction from a parameterized problem A to a parameterized problem Π , and Π is FPT, then A is FPT.*

The previous theorem shows that parameterized reductions work as intended. It says that

if A and Π are parameterized problems, A is W[1]-hard, and there is a parameterized reduction from A to Π , then Π is W[1]-hard.

2.3 CNF-SAT and Exponential Time Hypotheses

Let $X = \{x_1, \dots, x_n\}$ be a set of binary variables, taking values `true`, `false`. A *literal* is either a positive (x_i) or negated (\bar{x}_i) occurrence of a variable x_i . A *boolean formula in conjunctive normal form* or simply *CNF-formula* over X is an AND of ORs (called the *clauses*) of literals from X . For instance,³ the following is a CNF-formula over $\{x_1, x_2, x_3\}$:

$$\varphi = (x_1 \vee \bar{x}_2) \wedge (x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \quad (2.1)$$

One of the most fundamental problems in computer science is CNF-SAT, which given a CNF-formula asks whether there is a truth assignment to its variables that *satisfies* each of its clauses, meaning that each of its clauses evaluates to `true`. In the above example (2.1), if we set $x_1 = \text{false}$, $x_2 = \text{false}$ and $x_3 = \text{true}$, then all clauses of φ are satisfied. On the other hand, if we forbid this truth assignment by adding a clause $(x_1 \vee x_2 \vee \bar{x}_3)$, then the resulting formula has no satisfying assignment.

For a positive integer q , the q -SAT problem is the restriction of CNF-SAT to CNF-formulas whose clauses contain at most q literals.

Not-All-Equal SAT. Given a CNF-formula φ , a *not-all-equal assignment* of its variables is a truth assignment such that in each clause, at least one literal evaluates to `true` and at least one literal evaluates to `false`. For instance, if φ' is the CNF-formula obtained from φ given in (2.1) by replacing the last literal in the last clause with \bar{x}_3 , then the assignment $x_1 = \text{false}$, $x_2 = \text{false}$ and $x_3 = \text{true}$ is a not-all-equal assignment for φ' .

²Using the standard assumption that f and g are nondecreasing.

³This example is taken from [190, page 1].

In the corresponding NOT-ALL-EQUAL SAT problem (NAE-SAT for short), we are given a CNF-formula and the question is whether it has a not-all-equal assignment. Again, for a constant q , the q -NAE-SAT problem is the restriction of NAE-SAT to formulas with maximum clause length q .

Exponential Time Hypotheses. In 2001, Impagliazzo et al. [162, 163] made two conjectures about the complexity of q -SAT. These conjectures are known as the Exponential Time Hypothesis (ETH) and Strong Exponential Time Hypothesis (SETH), which can be stated as shown below. For the exact statement and a thorough introduction to the resulting conditional hardness theory, we refer to the book [93], and for surveys of conditional lower bounds based on such conjectures, see e.g. [207, 280].

Conjecture 2.4 (ETH [162]). There is an $\epsilon > 0$, such that 3-SAT on n variables cannot be solved in time $\mathcal{O}^(2^{\epsilon n})$.*

Conjecture 2.5 (SETH [162, 163]). For every $\epsilon > 0$, there is a $q \in \mathcal{O}(1)$ such that q -SAT on n variables cannot be solved in time $\mathcal{O}^((2 - \epsilon)^n)$.*

3

Width Measures

This chapter is meant to give a brief introduction to the three width measures that are most relevant for the remainder of this thesis: treewidth, clique-width, and mim-width. Besides the basic definitions and some illustrations, we discuss in each case one property of decompositions of small width that is useful in algorithmic applications. To showcase bounded-width decompositions ‘in action’, we describe an algorithm for the **MAXIMUM INDEPENDENT SET** problem parameterized by each of the three width measures. Recall that an *independent set* of a graph is a set of pairwise non-adjacent vertices. The following problem will be the playground for this chapter.

MAXIMUM INDEPENDENT SET

Input: A graph G .

Question: What is the size of a maximum independent set in G ?

The order in which we present the width measures is according to increasing power of expressive strength, and in the latter two cases we give short comparisons with the direct predecessor in this order towards the end of each respective section. Moreover, as all algorithms we present here (and throughout this thesis) always rely on having some decomposition of small width at hand, we give brief overviews on the state of the art of algorithms computing bounded-width decompositions.

3.1 Treewidth and Pathwidth

The first width measure we introduce here is that of *treewidth* which is arguably the most widely used and deeply understood one [93, 105]. Intuitively speaking, the treewidth of a graph G measures how far away you have to step back from G until it looks like a forest, or, if G is connected, like a tree. From an algorithmic point of view, graphs of small treewidth can be understood as graphs that can be decomposed

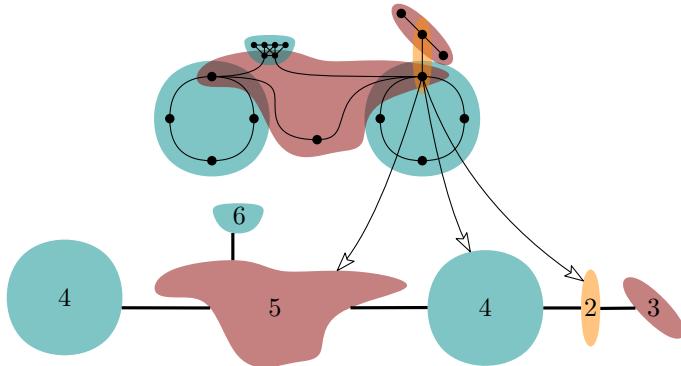


Figure 3.1: An example of a graph with one of its tree decompositions of width 5. The arrows show that the tree decomposition satisfies property T3 for that vertex. Note that replacing the bags of size 5 and 6 by a single bag containing the union of all these vertices results in a path decomposition of width 8. The treewidth of this graph is in fact 3.

in a tree-like way along small separators. The analogy with trees is apparent: in a tree, the removal of any edge disconnects the graph, and this is the main reason why many computational problems that are hard on general graphs can be solved in polynomial time on trees.

Definition 3.1 (Treewidth, Pathwidth). Let G be a graph. A *tree decomposition* of G is a pair (T, \mathcal{B}) of a tree T and an indexed family of vertex subsets $\mathcal{B} = \{B_t \subseteq V(G)\}_{t \in V(T)}$, called *bags*, satisfying the following properties.

- (T1) $\bigcup_{t \in V(T)} B_t = V(G)$.
 - (T2) For each $uv \in E(G)$ there exists some $t \in V(T)$ such that $\{u, v\} \subseteq B_t$.
 - (T3) For each $v \in V(G)$, let $U_v := \{t \in V(T) \mid v \in B_t\}$ be the nodes in T whose bag contains v . Then, $T[U_v]$ is connected.

The *width* of (T, \mathcal{B}) is $\max_{t \in V(T)} |B_t| - 1$, and the *treewidth* of a graph G , often denoted by $\text{tw}(G)$, is the minimum width over all tree decompositions of G .

If T is a path, then (T, \mathcal{B}) is called a *path decomposition*, and the *pathwidth* of a graph is the minimum width over all its path decompositions.

To avoid confusion, we usually refer to the vertices of the tree in a tree decomposition as *nodes*. Since the definition of treewidth looks rather technical, we give an illustration in Figure 3.1. Let us briefly argue the *separator property* of tree decompositions, saying that in a tree decomposition (T, \mathcal{B}) of a graph G ,

for every edge $st \in E(T)$, $B_s \cap B_t$ is a separator of G , (3.1)

as alluded to above. Let T_s be the component of $T - st$ containing s , and T_t the component of $T - st$ containing t . We argue that every vertex $u \in V_s := \bigcup_{p \in V(T_s)} B_p$ that has a neighbor $v \in V(G - V_s)$ is contained in $B_s \cap B_t$, which implies the claim. Since u and v are adjacent we know by property T2 that there is some node $t^* \in V(T)$ such that $\{u, v\} \subseteq B_{t^*}$. Moreover, since v is not contained in V_s , we know that $t^* \in V(T_t)$. On the other hand, we know that there is some node $s^* \in V(T_s)$ whose bag contains u . Since both B_{s^*} and B_{t^*} contain u , the bags at all nodes on the path from s^* to t^* in T contain u by property T3, in particular we have that B_s and B_t both contain u .

Now that we have argued the most basic algorithmically useful property of tree decompositions, we introduce a special type of tree decomposition that allows for streamlining the description of dynamic programming algorithms along tree decompositions. We will give an example of such an algorithm below.

Definition 3.2 (Nice Tree Decomposition). Let G be a graph and (T, \mathcal{B}) a tree decomposition of G . Then, (T, \mathcal{B}) is called a *nice tree decomposition*, if T is rooted and each node is of one of the following types.

Leaf. A node $t \in V(T)$ is a *leaf node*, if t is a leaf of T and $B_t = \emptyset$.

Introduce. A node $t \in V(T)$ is an *introduce node* if it has precisely one child s , and there is a unique vertex $v \in V(G) \setminus B_s$ such that $B_t = B_s \cup \{v\}$. In this case we say that v is *introduced at t* .

Forget. A node $t \in V(T)$ is a *forget node*, if it has precisely one child s , and there is a unique vertex $v \in B_s$ such that $B_t = B_s \setminus \{v\}$. In this case we say that v is *forgotten at t* .

Join. A node $t \in V(T)$ is a *join node*, if it has precisely two children s_1 and s_2 , and $B_t = B_{s_1} = B_{s_2}$.

For each $t \in V(T)$, let T_t denote the subtree of T rooted at t ; we let $V_t := \bigcup_{t \in V(T_t)} B_t$ and $G_t := G[V_t]$.

It is known that any tree decomposition of a graph can be transformed in polynomial time into a nice tree decomposition with a relatively small number of bags.

Lemma 3.3 (Kloks [187]). *Let G be a graph on n vertices, and let k be a positive integer. Any width- k tree decomposition (T, \mathcal{B}) of G can be transformed in time $\mathcal{O}(k \cdot |V(T)|)$ into a nice tree decomposition (T', \mathcal{B}') of width k such that $|V(T')| = \mathcal{O}(k \cdot n)$.*

Note that the bound on the number of nodes in the nice tree decomposition of the above lemma is essentially tight [93, Exercise 7.3].

Maximum Independent Set

We now exemplify the use of (nice) tree decompositions in algorithms. In particular, we consider the MAXIMUM INDEPENDENT SET problem, in the setting that we are given a tree decomposition of bounded width at the input. Formally, we deal with the following parameterized problem. Recall that a vertex set $S \subseteq V(G)$ in a graph G is called an *independent set* if there is no edge between any pair of vertices in S , and that an independent set S is a *maximum* independent set if for any other independent set S' of G , $|S| \geq |S'|$.

MAXIMUM INDEPENDENT SET parameterized by the width tw of (T, \mathcal{B}) .

Input: Graph G with one of its tree decompositions (T, \mathcal{B}) .

Question: What is the size of a maximum independent set in G ?

The algorithm we design below is bottom-up dynamic programming along the (rooted) tree T of a nice tree decomposition. It resembles the process of building independent sets of the input graph G during the bottom-up traversal of T . This means that at each node $t \in V(T)$, we have to store information about independent sets in G_t , which is where the structure of tree decompositions is being exploited. Any independent set in G is obtained from an independent set I in G_t by adding vertices from $G - V_t$. But the only information that is relevant for solving MAXIMUM INDEPENDENT SET is the *intersection of I with the top bag B_t* . By the separator property, the vertices in $I \setminus B_t$ have no neighbors in $G - V_t$. Therefore, if we have another independent set J in G_t with $J \cap B_t = I \cap B_t$, it suffices to remember the size of the larger one of the two independent sets: For each set $A \subseteq V(G) \setminus V_t$ ‘from the outside’, we have that $I \cup A$ is an independent set in G if and only if $J \cup A$ is. We now show how to turn this observation into an algorithm.

Example 3.4 (Maximum Independent Set/tw). First, we apply Lemma 3.3 to transform the given tree decomposition at the input into a nice tree decomposition (T, \mathcal{B}) .

We give a dynamic programming algorithm that stores at each node $t \in V(T)$, and for each subset $S \subseteq B_t$, the size of a maximum independent set in G_t whose intersection with B_t is equal to S . Formally, for each $t \in V(T)$ and each $S \subseteq B_t$, we store a table entry

$$\text{tab}[t, S] = \max\{|X| : X \text{ is an independent set in } G_t \text{ and } X \cap B_t = S\}.$$

Note that the previous definition immediately requires S itself to be an independent set. As a convention, if S is not independent, we set $\text{tab}[t, S] = -\infty$.

Below, we show how to compute the table entries at each type of node in the nice tree decomposition, assuming that the table entries at the children (if any) have been computed. The algorithm then traverses the tree decomposition in a bottom-up

manner, starting at the leaf nodes, and follows the description given below at each node. Once the table entries at the root have been computed, we can output the solution to the instance at hand: Let $\mathbf{r} \in V(T)$ be the root node of T . Since $G_{\mathbf{r}} = G$, for each $S \subseteq B_{\mathbf{r}}$, the table entry $\text{tab}[\mathbf{r}, S]$ stores the size of the largest independent set in G whose intersection with $B_{\mathbf{r}}$ is equal to S . The solution to the instance at hand is therefore the maximum value of all table entries computed for the root \mathbf{r} . Now let $t \in V(T)$, and assume that the table entry at its children (if any) have been computed.

Leaf. If t is a leaf node, then $B_t = \emptyset$. In this case, G_t has no vertices, and the only table entry we have to store is $\text{tab}[t, \emptyset] = 0$.

Forget. Suppose that t is a forget node with child s , and let v be the vertex forgotten at t . We can observe that $G_t = G_s$ which means that the independent sets of G_t and the ones of G_s coincide. For each $S \subseteq B_t$, the only thing we have to take into account is that an independent set whose intersection with B_t is S may or may not contain the vertex v . Therefore,

$$\text{tab}[t, S] := \max\{\text{tab}[s, S], \text{tab}[s, S \cup \{v\}]\}.$$

Introduce. Now suppose that t is an introduce node with child s , and let v be the vertex introduced at t . For any $S \subseteq B_t$, we do the following. If $v \notin S$, then we can simply look up the corresponding table entry at node s . (The only difference between G_t and G_s is the vertex v and its neighbors in G_s .) If $v \in S$, then two things can happen. If v has a neighbor in S , then clearly S is not an independent set, so the table entry should be $-\infty$. If v has no neighbor in S , then the maximum independent in G_t whose intersection with B_t is S is obtained from the maximum independent set I_s^* in G_s whose intersection with B_s is $S \setminus \{v\}$, by adding v : we have that $V_t = V_s \cup \{v\}$, and by the separator property (3.1), v has no neighbors in $I_s^* \setminus B_s$, so $I_s^* \cup \{v\}$ is an independent set. To summarize, for each $S \subseteq B_t$, we let

$$\text{tab}[t, S] := \begin{cases} \text{tab}[s, S], & \text{if } v \notin S \\ \text{tab}[s, S \setminus \{v\}] + 1, & \text{if } v \in S \text{ and } N(v) \cap S = \emptyset \\ -\infty, & \text{otherwise} \end{cases}$$

(Note that in the ‘otherwise’ case, we have that $v \in S$ and v has a neighbor in S , so S is not an independent set.)

Join. If t is a join node with children s_1 and s_2 , then we have that $B_t = B_{s_1} = B_{s_2}$. Moreover, since there are no edges between $G_{s_1} - B_t$ and $G_{s_2} - B_t$, each independent set of G_t can be obtained from an independent set of G_{s_1} and one of G_{s_2} such that they have the same intersection with the vertices in B_t . In other words, for any $S \subseteq B_t$, the largest independent set in G_t

whose intersection with B_t is equal to S is obtained from the union of the largest independent set in G_{s_1} whose intersection with B_{s_1} is S and largest independent set in G_{s_2} whose intersection with B_{s_2} is S . Therefore, for each $S \subseteq B_t$, we let $\text{tab}[t, S] := \text{tab}[s_1, S] + \text{tab}[s_2, S]$.

From the above description, it is clear that for each node $t \in V(T)$ and each of the at most $2^{\text{tw}+1}$ subsets S of B_t , the computation of the table entry $\text{tab}[t, S]$ takes time at most $\mathcal{O}(\text{tw})$. (The bottleneck is at the introduce node, when we have to check if v has neighbors in S .) By Lemma 3.3, $|V(T)| = \mathcal{O}(\text{tw} \cdot n)$. Under the (relatively harmless) assumption that the tree decomposition that we initially received at the input had $n^{\mathcal{O}(1)}$ nodes as well, the total runtime of the algorithm is $\mathcal{O}(2^{\text{tw}} \cdot \text{tw}^2 \cdot n) + n^{\mathcal{O}(1)} = \mathcal{O}^*(2^{\text{tw}})$. \triangleleft

We would like to remark that even though the above algorithm is very elementary, it is likely to be asymptotically optimal. Lokshtanov et al. [208] showed that unless the Strong Exponential Time Hypothesis (see Section 2.3) fails, there is no $\epsilon > 0$ such that MAXIMUM INDEPENDENT SET can be solved in time $\mathcal{O}^*((2 - \epsilon)^{\text{tw}})$ in this setting.

3.1.1 Computing Treewidth

Treewidth-based algorithms typically assume that a tree decomposition of small width is provided at the input; constructing such decompositions, however, is anything but trivial. We briefly discuss some results on this problem, formally stated as follows.

TREE DECOMPOSITION

- | | |
|---------------|---|
| <i>Input:</i> | Graph G (on n vertices), integer k |
| <i>Task:</i> | Construct a tree decomposition of G of width at most k , if it exists; report $\text{tw}(G) > k$, otherwise. |

Arnborg et al. [6] showed that *deciding* if a given graph has treewidth at most k is NP-complete. In terms of polynomial-time approximation algorithms, the current best algorithm due to Feige et al. [118] constructs a tree decomposition of width $\mathcal{O}(k\sqrt{\log k})$. The existence of polynomial-time constant-factor approximation algorithms is considered unlikely, as Wu et al. [287] showed that under a common assumption in approximation algorithms¹ this task is NP-hard. The fastest known exact exponential-time algorithm to compute the treewidth of an n -vertex graph runs in time $\mathcal{O}^*(1.734601^n)$. This bound is due to Fomin et al. [129], improving upon a bound on enumerating potential maximal cliques due to Fomin and Villanger [130], combined with the algorithmic machinery based on minimal separators and potential

¹Concretely, the Small Set Expansion Conjecture [246], a relative of the Unique Games Conjecture.

maximal cliques by Fomin et al. [128]. Fomin and Villanger [130] also showed that there is an $\mathcal{O}^*(2.6151^n)$ time and *polynomial space* algorithm for TREE DECOMPOSITION.

In terms of parameterized algorithms, it was first observed by Arnborg et al. [6] that TREE DECOMPOSITION parameterized by k is in XP, more specifically that it has an $\mathcal{O}(n^{k+2})$ time algorithm. Arguably the most important types of algorithms for the parameterized algorithms community, however, are those whose running time is fixed-parameter tractable in k . Compared to *polynomial*-time approximation algorithms, this additional budget in the running time allows for constant-factor approximations. For instance, Robertson and Seymour [256] gave a (roughly) 4-approximation running in time $\mathcal{O}(27^k \cdot k^2 \cdot n^2)$. Several expositions of this classic result can be found in [93, 186, 245].

Many FPT-algorithms parameterized by treewidth (for instance, the algorithm for MAXIMUM INDEPENDENT SET we presented above) have a runtime whose dependence on the number of vertices n is *linear*. Applying the Robertson-Seymour approximation increases the overall dependence on n in the runtime to quadratic. The celebrated *exact* algorithm due to Bodlaender [29], running in time $2^{\mathcal{O}(k^3)} \cdot n$, resolved this issue. We will see some of the far-reaching consequences of this result in Section 3.5.

On the downside, the runtime of Bodlaender's algorithm has a quite heavy dependence on k , which surpasses the ($f(k)$ -part of the) runtime of many common tree decomposition based FPT-algorithms. (For instance, for the MAXIMUM INDEPENDENT SET algorithm above, $f(k) = 2^k \cdot k^{\mathcal{O}(1)}$.) A desirable goal is to have algorithms whose running time is *single-exponential* in the treewidth k and linear in n , meaning functions of the form $2^{\mathcal{O}(k)} \cdot n$. In such applications, a constant-factor approximation for treewidth suffices. Tailored to such cases, Bodlaender et al. [34] gave a $2^{\mathcal{O}(k)} \cdot n$ -time 5-approximation algorithm for TREE DECOMPOSITION.

To this day, more than two decades after the publication of Bodlaender's algorithm, there is still no exact FPT-algorithm for TREE DECOMPOSITION running in time $2^{o(k^3)} \cdot n^{\mathcal{O}(1)}$ (note that we relaxed the requirement on the dependence on n to be polynomial). The existence of such algorithms is a prominent and long-standing open problem in the field [32]. In Chapter 6, we successfully address a runtime bottleneck of Bodlaender's algorithm. However, on its own, it does not yet lead to an asymptotic improvement. We believe that it could open up a new route towards such an algorithm.

Open Problem 3.1. Is there an algorithm for TREE DECOMPOSITION running in time $2^{o(k^3)} \cdot n^{\mathcal{O}(1)}$?

Historic Notes

We end this brief introduction to treewidth on some historic notes. While equivalent notions have been defined earlier in the literature [23, 153], the definition we presented

here goes back to the work of Robertson and Seymour [253, 254]. Rather than an algorithmic tool, treewidth was introduced as a stepping stone towards a giant among all theorems in graph theory, whose proof easily spans 500 pages: namely that in each infinite set of graphs, there is one graph that is a minor of another. This theorem is known as the *Graph Minor Theorem*, and considered the resolution of a conjecture often attributed to Wagner [284].² We refer to [265] for an interesting anecdotal account due to Paul Seymour of how treewidth came into play in the proof of the Graph Minor Theorem.

3.2 Branch Decompositions

Several width measures that we will encounter in the remainder of this chapter are based on *branch decompositions*, a concept we introduce in this section. Informally speaking, a branch decomposition of a graph G represents a way of recursively splitting the vertex set of a graph into two parts A and B that comes together with a guarantee on the structure of the graph induced by the edges *crossing from A to B* , i.e. the subgraph of G containing all edges with one endpoint in A and the other endpoint in B , denoted by $G[A, B]$. This guarantee is the (numerical) *width* of the branch decomposition, and typically asserts that whenever the width is *small*, then the structure of $G[A, B]$ is *simple*. The concrete meaning of ‘simple’ depends on the width measure at hand; for instance, in the example below we consider carving-width, which counts the number of edges in $G[A, B]$. Another, more complicated, example is mim-width which we introduce in Section 3.4; there, the measure is the maximum size of any induced matching in $G[A, B]$.

Abstractly, we can model such a width measure as a function from subsets of $V(G)$ to the non-negative real numbers. (Most width measures are in fact integral but for the sake of generality, we stick with real numbers for the moment.) For instance, carving-width can be expressed as the function carw_G , where for each $A \subseteq V(G)$, the value $\text{carw}_G(A)$ is the number of edges with one endpoint inside A and the other endpoint outside A . Such functions f_G (including carw_G) are often symmetric, meaning that for all $A \subseteq V(G)$, $f_G(A) = f_G(V(G) \setminus A)$. Asymmetric functions are usually considered in the context of *rooted* branch decompositions, where the recursive splitting has some sense of direction. That is, together with the two parts A and B , we get the information that A is the *child* of B , in which case we can choose to evaluate the width function only at the set A . For instance, in Section 3.3, we encounter *module-width*; denoting the corresponding set function by mwval_G , there may be some $A \subseteq V(G)$ with $\text{mwval}_G(A) = k$ and $\text{mwval}_G(V(G) \setminus A) = 2^k$.

Before we give the formal definition, we discuss how we model the ‘(rooted) recursive splitting’ mentioned above. We consider a (rooted) tree T of maximum

²However, “Wagner himself always insisted he did not [conjecture it]—even after the graph minor theorem had been proved” [101, p. 388].

degree three together with a bijection from the vertices of G to the leaves of T (denoted by $L(T)$ throughout). Each edge $e \in E(T)$ of this tree naturally splits $V(G)$ into two parts: $T - e$ has two connected components, say T_A and T_B , and we obtain A and B as the set of vertices that are mapped to a leaf in T_A and T_B , respectively. In this case, we say that (A, B) is the *cut associated with the edge e*. If T is rooted, then the edge $e = pt$ is such that p is the parent of t . In this case we may take the set A as the vertices mapped to leaves in the component of $T - e$ containing t , the child.

Definition 3.5 (Branch Decomposition). Let G be a graph. A *branch decomposition* is a pair (T, \mathcal{L}) of a subcubic tree T and a bijection $\mathcal{L}: V(G) \rightarrow L(T)$. If T is rooted, then we call (T, \mathcal{L}) a *rooted branch decomposition*.

- For any subtree T' of T , we denote by $A(T')$ the set of vertices of G that \mathcal{L} maps to the leaves of T' , i.e. $A(T') := \mathcal{L}^{-1}(L(T'))$.
- If T is rooted, then for each $t \in V(T)$, we let T_t denote the subtree of T rooted at t , and we let $V_t := A(T_t)$, $G_t := G[V_t]$, and $\bar{V}_t := V(G) \setminus V_t$.

Given a set function $f: 2^{V(G)} \rightarrow \mathbb{R}_{\geq 0}$, we want to arrive at a notion of *f-width* of a branch decomposition, and finally at the *f-width* of G . The *f-width* of a branch decomposition (T, \mathcal{L}) is the maximum value of any evaluation of f over all subsets that are encountered at an edge of T ; and the *f-width* of G is the minimum *f-width* over all branch decompositions of G . We first give the definition for unrooted branch decompositions. Recall that for a graph H , $cc(H)$ denotes the set of connected components of H .

Definition 3.6 (f-Width). Let G be a graph and $f: 2^{V(G)} \rightarrow \mathbb{R}_{\geq 0}$ a set function. Let furthermore (T, \mathcal{L}) be an (unrooted) branch decomposition of G . The *f-width* of (T, \mathcal{L}) is

$$\max_{e \in E(T), T' \in cc(T-e)} f(A(T')).$$

The *f-width* of G is the minimum *f-width* over all branch decompositions of G .

The rooted *f-width* is defined analogously, additionally taking into account the ancestral relationship imposed by the root of the branch decomposition.

Definition 3.7 (Rooted f-Width). Let G be a graph and $f: 2^{V(G)} \rightarrow \mathbb{R}_{\geq 0}$ a set function. Let furthermore (T, \mathcal{L}) be a rooted branch decomposition of G . The *f-width* of (T, \mathcal{L}) is

$$\max_{t \in V(T)} f(V_t).$$

The *rooted f-width* of G is the minimum *f-width* over all rooted branch decompositions of G .

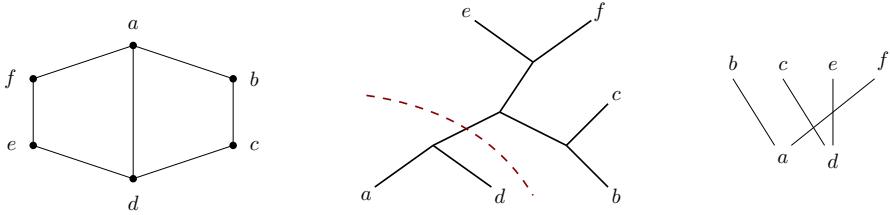


Figure 3.2: A graph G on the left, one of its branch decompositions of carving-width 4 in the middle, and on the right the bipartite subgraph of G associated with the depicted cut. Observe that four edges cross this cut, and that this is the maximum in this branch decomposition.

Observe that for a symmetric set function f , the f -width of G and rooted f -width of G coincide. We illustrate the concept of f -width in the following example. While the width measure carving-width is not of importance to the rest of this thesis, it illustrates the concept of a branch decomposition quite well. It is interesting to observe that graphs of bounded carving-width are graphs of bounded treewidth and bounded maximum degree, concretely, it is known that the carving-width of any graph G is at most $(\text{tw}(G) + 1) \cdot \Delta(G)$ [231].

Example 3.8 (Carving-Width). Let G be a graph, and $\text{carw}_G: 2^{V(G)} \rightarrow \mathbb{N}$ be the set function where for all $A \subseteq V(G)$,

$$\text{carw}_G(A) = |\{ab \in E(G) \mid a \in A \wedge b \notin A\}|.$$

The *carving-width* of G is the carw_G -width of G . We give an example of a graph with one of its branch decompositions of carw_G -width 4 in Figure 3.2.

Linear Branch Decompositions

In the same way that pathwidth can be considered a ‘linear variant of treewidth’, width measures defined in terms of branch decompositions can be restricted to linear decompositions. A *linear* branch decomposition of a graph G is a branch decomposition (T, \mathcal{L}) , where T is a *caterpillar*. Recall that if T is a caterpillar, then T has a subgraph P that is a path, and each vertex of $V(T) \setminus V(P)$ is only adjacent to a single vertex of P . Suppose wlog. that P is maximal. In a *rooted* version of (T, \mathcal{L}) , we always assume that the root of T is one of the two endpoints of P . The (*rooted*) *linear f -width* of a graph G is then defined as the f -width where we take the minimum width over all (*rooted*) linear branch decompositions.

Suppose (T, \mathcal{L}) is a rooted linear branch decomposition and let P denote the (maximal) path inside T . Since T is subcubic, each internal node of P is adjacent to precisely one leaf, and since P is maximal, its endpoints are leaves of T . The root r of

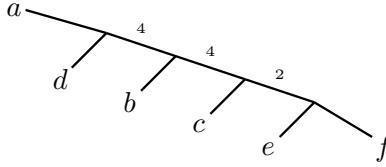


Figure 3.3: A *linear* branch decomposition of the graph depicted in Figure 3.2. Note that the carving-width of this decomposition is as well 4, with the small numbers on the edges giving the **carw**-value of their cuts. (The value of a cut at an edge incident with a leaf is equal to the degree of the vertex mapped to the leaf.) This linear branch decomposition, if rooted at the node to which a is mapped, loosely corresponds to the linear order f, e, c, b, d, a .

T is one of the endpoints of P . The other endpoint of P is the lowest leaf of T , say z . Therefore, traversing T in a bottom-up manner, starting at z , naturally corresponds to a linear order of the vertices of G , see Figure 3.3 for an illustration. For several³ width measures, cuts that have a single vertex on one side exhibit low complexity; in such cases, considering only the cuts in such a linear order is practically equivalent to considering the linear (rooted) branch decomposition.

Computing Optimal Branch Decompositions

Suppose we want to find an optimal (rooted) branch decomposition of an n -vertex graph minimizing *some* width function f . For simplicity, we make the assumption that f does not take on any value that is larger than $2^{\mathcal{O}(n)}$, and let **eval** denote the maximum time it takes to evaluate f at any vertex subset. Trying all possible (rooted) branch decompositions essentially means trying all full binary trees with n leaves, in other words, all full binary trees on $2n - 1$ nodes. The number of such trees is the $(2n - 1)$ -th *Catalan number*, whose value is equal to

$$\frac{1}{2n} \binom{2(2n - 1)}{2n - 1} = 2^{\Theta(n \log n)}.$$

Therefore, using this brute force approach we only obtain an algorithm whose runtime is not single-exponential, i.e. not of the form $\mathcal{O}^*(2^{\mathcal{O}(n)} \cdot \text{eval})$. As observed by Oum [237], an $\mathcal{O}^*(3^n \cdot \text{eval})$ time algorithm can easily be obtained via dynamic programming over subsets. Oum [237] furthermore gave an algorithm that improves this to $\mathcal{O}^*(2^n \cdot \text{eval})$.

³Observe that this is *not* the case for a linear variant of carving width: take for instance a star, and a linear branch decomposition that places its center roughly in the middle of the ordering associated with the decomposition.

Branch Decompositions vs Branch Decompositions vs ...

We briefly discuss the usage of the term ‘branch decomposition’ throughout this text. Originally, branch decompositions and the corresponding width measure *branch-width* were introduced by Robertson and Seymour [255] as a close relative to treewidth. In this original definition, the *edges* rather than the vertices of a graph G were mapped bijectively to the leaves of a subcubic tree T ; the width of a branch decomposition (in this sense) is the maximum, over all edges e of T , of the number of vertices appearing as endpoints of edges in both sides of the cut associated with e . The *branchwidth* of a graph G , $\text{bw}(G)$, is the minimum width of any such branch decomposition. Robertson and Seymour [255] showed that for any graph G of branch-width greater than 1,

$$\text{bw}(G) - 1 \leq \text{tw}(G) \leq \left\lfloor \frac{3}{2} \cdot \text{bw}(G) \right\rfloor - 1.$$

In the same work, Robertson and Seymour also observed that branch decompositions can be generalized to matroids [255, Section 12], in which case the elements of the ground set are mapped to the leaves of the branch decomposition. This notion has received considerable attention since, see e.g. [140, 159]. Moreover, in [240], Oum and Seymour consider branch decompositions of *symmetric submodular set functions*, including functions over the vertex set of a graph.

In this work, we stick with the term ‘(rooted) branch decomposition’, taken over the vertex set of a graph, and we note that in related works, it is sometimes replaced with terms such as ‘(rooted) layout’ or ‘(rooted) decomposition tree’.

3.3 Clique-Width and Module-Width

The notion of clique-width has been introduced by Courcelle, Engelfriet, and Rozenberg [87], and is a width measure that is strictly more powerful than treewidth. This means that any graph of bounded treewidth has bounded clique-width, but not vice versa. (We give some precise bounds in Section 3.3.1.) Clique-width is defined in terms of a set of operations on *labeled* graphs, defined as follows.

Let Ω be a set. An Ω -*labeled graph* is a pair (G, λ) , where $\lambda: V(G) \rightarrow \Omega$ is a map assigning each vertex of G a *label* from Ω . For each $i \in \Omega$, we call the set of vertices with label i , i.e. the set $\{v \in V(G) \mid \lambda(v) = i\}$, the *label class* i . For a positive integer w , we use the shorthand ‘ w -labeled graph’ for ‘ $[w]$ -labeled graph’.

Now, the clique-width- w operations can be used to inductively construct several types of w -labeled graphs. The most fundamental operation is to create a new labeled graph on a single vertex, with some label $i \in [w]$ (denoted by ‘ \bullet_i ’). The remaining operations are: taking the disjoint union of two w -labeled graphs (‘ \oplus ’), for $i \neq j$, making all vertices labeled i adjacent to all vertices labeled j (‘ $\eta_{i,j}$ ’), and again for $i \neq j$, reassigning all vertices labeled i the label j (‘ $\rho_{i \rightarrow j}$ ’). We give the definition

in two stages: first, we describe how to form expressions using the above operators, and second, we describe how to evaluate such expressions to labeled graphs.

Definition 3.9. Let $w \in \mathbb{N}$. A *clique-width w-expression*, or simply w -expression is a string obtained inductively as follows:

- (i) For all $i \in [w]$, ‘ \bullet_i ’ is a w -expression.
- (ii) If φ_1 and φ_2 are w -expressions, then ‘ $\varphi_1 \oplus \varphi_2$ ’ is a w -expression.
- (iii) For all $i, j \in [w]$, $i \neq j$, if φ is a w -expression, then ‘ $\eta_{i,j}(\varphi)$ ’ is a w -expression.
- (iv) For all $i, j \in [w]$, $i \neq j$, if φ is a w -expression, then ‘ $\rho_{i \rightarrow j}(\varphi)$ ’ is a w -expression.

An *evaluation* of a w -expression φ is a w -labeled graph, denoted by $\text{eval}(\varphi)$, inductively obtained as follows.

- (I) If $\varphi = \bullet_i$, then $\text{eval}(\varphi)$ consists of a single vertex whose label is i .
- (II) If $\varphi = (\varphi_1 \oplus \varphi_2)$, then $\text{eval}(\varphi) = \text{eval}(\varphi_1) \dot{\cup} \text{eval}(\varphi_2)$, where ‘ $\dot{\cup}$ ’ denotes the disjoint union.
- (III) If $\varphi = \eta_{i,j}(\varphi')$, then $\text{eval}(\varphi)$ is obtained from $\text{eval}(\varphi')$ by making all vertices labeled i adjacent to all vertices labeled j .
- (IV) If $\varphi = \rho_{i \rightarrow j}(\varphi')$, then $\text{eval}(\varphi)$ is obtained from $\text{eval}(\varphi')$ by assigning all vertices labeled i the label j .

The *clique-width* of a graph G is the minimum integer w such that there is a w -expression φ with $\text{eval}(\varphi) = (G, \lambda)$.

Note that (an evaluation of) a w -expression φ can be represented by a subcubic rooted node-labeled tree τ , called *syntactic tree*, in the following way. If, as in (I), $\varphi = \bullet_i$, then τ consists of a single node, with the single vertex of $\text{eval}(\varphi)$ mapped to it. If, as in (II), $\varphi = \varphi_1 \oplus \varphi_2$, then let by induction τ_1 and τ_2 be the syntactic tree of $\text{eval}(\varphi_1)$ and of $\text{eval}(\varphi_2)$, respectively. In this case, τ is obtained by a new node r labeled ‘ \oplus ’ which will be the root of τ , and whose two children are the roots of τ_1 and τ_2 . Similarly, if, as in (III), $\varphi = \eta_{i,j}(\varphi')$, or, as in (IV), $\varphi = \rho_{i \rightarrow j}(\varphi')$, then let τ' be the syntactic tree of φ' . In both cases, τ is obtained from adding a new node r labeled ‘ $\eta_{i,j}$ ’ or ‘ $\rho_{i \rightarrow j}$ ’ (depending on which case we are in) as a new root node, and making r the parent of the root node of τ' .

Let τ be a syntactic tree. With each node of $t \in V(\tau)$, we associate the labeled graph G_t which is the evaluation of the w -expression represented by the subtree of τ rooted at t . The following property is crucially used in many structural and algorithmic results about clique-width, and is furthermore instructive in the exposition of an alternative way of viewing clique-width given below.

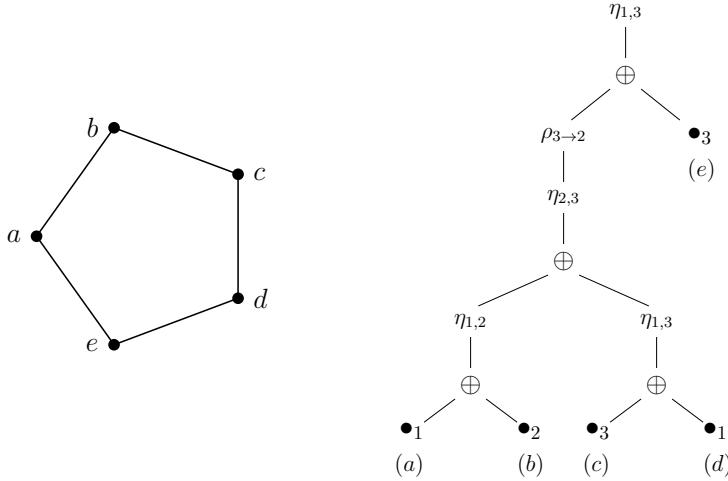


Figure 3.4: An illustration of C_5 together with the syntactic tree of the 3-expression given in Equation (3.2) that evaluates to (a labeled version of) C_5 .

Observation 3.10. Let $w \in \mathbb{N}$, and let τ be a syntactic tree of a w -expression evaluating to a labeled graph (G, λ) . Let $t \in V(\tau)$, and let $u, v \in V(G_t)$ be two vertices with $\lambda(u) = \lambda(v)$. Then, $N_G(u) \cap (V(G) \setminus V(G_t)) = N_G(v) \cap (V(G) \setminus V(G_t))$, i.e. u and v are twins w.r.t. the vertices outside of G_t .

The previous observation is justified as follows. Once two vertices are assigned the same label, they remain in the same label class as the evaluation continues along the syntactic tree. This directly implies that they receive the same set of neighbors as the evaluation progresses.

Example 3.11. To illustrate the notions of w -expressions and syntactic trees, we show how to construct C_5 from a 3-expression. In particular, the following 3-expression evaluates to (a labeled version of) C_5 :

$$\eta_{1,3} (\rho_{3 \rightarrow 2} (\eta_{2,3} (\eta_{1,2} (\bullet_1 \oplus \bullet_2) \oplus \eta_{1,3} (\bullet_3 \oplus \bullet_1))) \oplus \bullet_3) \quad (3.2)$$

We give an illustration of C_5 together with the syntactic tree corresponding to its 3-expression given in Equation (3.2) in Figure 3.4. For an illustration of the property described in Observation 3.10, consider the smallest subtree of the syntactic tree containing the vertices a and d . Observe that e is the only vertex that is not contained in that subtree. Moreover, both a and d have label 1, and they are both adjacent to e . \triangleleft

Several algorithmic results regarding clique-width in this work are given in terms of the (as we see below, equivalent) width-measure *module-width*, which we briefly

introduce now. Module-width is attributed to Rao [249, 250],⁴ and based on the notion of a rooted branch decomposition. On a high level, the module-width of a rooted branch decomposition bounds, at each of its nodes t , the number of subsets of $\overline{V_t}$ that make up the intersection of $\overline{V_t}$ with the neighborhood of some vertex in V_t . This is expressed in terms of an equivalence relation over V_t ; recall that for an equivalence relation \sim over a set Ω , we denote by Ω/\sim the set of equivalence classes of \sim and for each $x \in \Omega$, by $[x]_\sim$ the equivalence class of x .

Definition 3.12 (Module-Width). Let G be a graph, and (T, \mathcal{L}) be a rooted branch decomposition of G . For each $t \in V(T)$, let \sim_t be the equivalence relation on V_t defined as follows:

$$\forall u, v \in V_t : u \sim_t v \Leftrightarrow N_G(u) \cap \overline{V_t} = N_G(v) \cap \overline{V_t}$$

The *module-width* of (T, \mathcal{L}) is $\text{mw}(T, \mathcal{L}) := \max_{t \in V(T)} |V_t/\sim_t|$. The *module-width* of G , denoted by $\text{mw}(G)$, is the minimum module width over all rooted branch decompositions of G .

Remark 3.13. Using the framework introduced in Section 3.2, we could have defined module-width quickly as follows: ‘For a graph G , let $\text{mwval}_G : 2^{V(G)} \rightarrow \mathbb{N}$ be such that for all $A \subseteq V(G)$, $\text{mwval}_G(A) = |\{N(v) \cap (V(G) \setminus A) \mid v \in A\}|$. The module-width of G is the rooted mwval_G -width of G .’ However, the previous definition contains useful auxiliary notation that we use throughout this text, therefore we decided to give a direct definition.

From the above discussion about syntactic trees, the connection between clique-width and module-width is quite apparent: the shape of a syntactic tree is very similar to that of a rooted branch decomposition, and in a branch decomposition (T, \mathcal{L}) of module-width w , and each $t \in V(T)$, we can group the vertices of V_t according to the intersection of their neighborhoods with $\overline{V_t}$, to obtain at most w different groups. Viewing these as label classes, we have restored the crucial property about clique-width from Observation 3.10. Rao established the following bounds between clique-width and module-width.

Theorem 3.14 (Rao, Thm. 6.6 in [249]). *For any n -vertex graph G ,*

$$\text{mw}(G) \leq \text{cw}(G) \leq 2 \cdot \text{mw}(G).$$

Moreover, given a w -expression evaluating to G , a rooted branch decomposition of module-width w can be computed in time $\mathcal{O}(n^2)$; and given a rooted branch decomposition of G with module-width w , a $2w$ -expression evaluating to G can be computed in time $\mathcal{O}(n^2)$.

⁴Note that in [250], module-width is referred to as *modular-width* which usually has a different meaning, see e.g. [84].

Before we proceed with presenting the algorithm for MAXIMUM INDEPENDENT SET on graphs of bounded module-width, we introduce some more concepts related to rooted branch decompositions of bounded module-width. In particular, we describe at each internal node $t \in V(T)$ with children r and s an *operator* that will be useful when describing the dynamic programming algorithm. It tells us how the graph G_t is formed from G_r and G_s by disjoint union and addition of some edges, and how we can obtain the equivalence classes of \sim_t from the equivalence classes of \sim_r and \sim_s .

The operator (H_t, η_r, η_s) of node t with children r and s . First, it is clear that $V_t = V_r \cup V_s$. Since G_r and G_s are induced subgraphs of G_t , we furthermore know that $E(G_t[V_r]) = E(G_r)$ and $E(G_t[V_s]) = E(G_s)$, so it remains to describe the edges between V_r and V_s . By the definition of module-width, we know that each pair of vertices $u, v \in V_r$ with $u \sim_r v$ has the same neighborhood in $\overline{V_r} = V_s \cup \overline{V_t}$. Hence, for each vertex $z \in V_s$, we know that either both or neither of u and v are adjacent to z . In other words, for each pair $Q_r \in V_r/\sim_r$, $Q_s \in V_s/\sim_s$, either all edges between each pair of a vertex from Q_r and a vertex from Q_s are present in G_t , or none of them. This can be described by a bipartite graph H_t on bipartition $(V_r/\sim_r, V_s/\sim_s)$ such that:

$$\begin{aligned} E(G_t) &= E(G_r) \cup E(G_s) \cup F \text{ where} \\ F &= \{uv \mid u \in V_r, v \in V_s, \{[u]_{\sim_r}, [v]_{\sim_s}\} \in E(H_t)\} \end{aligned}$$

By roughly the same reasoning, we can observe that the equivalence relation \sim_t *coarsens* the equivalence relations \sim_r and \sim_s . Consider again vertices $u, v \in V_r$ such that $u \sim_r v$. Then, $N(u) \cap \overline{V_r} = N(v) \cap \overline{V_r}$, and since $V_r \subseteq V_t$ we have that $\overline{V_t} \subseteq \overline{V_r}$, which implies that $N(u) \cap \overline{V_t} = N(v) \cap \overline{V_t}$, so $u \sim_t v$. However, it may well be that there are vertices $u, v \in V_r$ with $u \not\sim_r v$, but $u \sim_t v$: this is the case when u and v have the same neighbors in $\overline{V_t}$, but different neighbors in V_s . Lastly, note that there may also be vertices $v \in V_r$ and $z \in V_s$ such that $v \sim_t z$.

We have argued that each equivalence class of \sim_t can be obtained by taking a subset of equivalence classes of \sim_r and \sim_s , and joining them (in what we call a ‘bubble’ below). Formally, there is a partition $\mathcal{P} = \{P_1, \dots, P_h\}$ of $V(H_t) = V_r/\sim_r \cup V_s/\sim_s$ such that $V_t/\sim_t = \{Q_1, \dots, Q_h\}$, where for $1 \leq i \leq h$, $Q_i = \bigcup_{Q \in P_i} Q$. For each $1 \leq i \leq h$, we call P_i the *bubble* of the resulting equivalence class $\bigcup_{Q \in P_i} Q$ of \sim_t . As auxiliary structures, for $p \in \{r, s\}$, we let $\eta_p: V_p/\sim_p \rightarrow V_t/\sim_t$ be the map such that for all $Q_p \in V_p/\sim_p$, $Q_p \subseteq \eta_p(Q_p)$, i.e. $\eta_p(Q_p)$ is the equivalence class of \sim_t whose bubble contains Q_p . We call (H_t, η_r, η_s) the *operator* of t , and illustrate this concept in Figure 3.5.

We introduce some more notation. For a node $t \in V(T)$ and a set $S \subseteq V(G_t)$, we let $\text{eqc}_t(S)$ be the set of all equivalence classes of \sim_t which have a nonempty intersection with S , and $\overline{\text{eqc}}_t(S)$ be the remaining equivalence classes of \sim_t . Formally, $\text{eqc}_t(S) := \{Q \in V_t/\sim_t \mid Q \cap S \neq \emptyset\}$ and $\overline{\text{eqc}}_t(S) := V_t/\sim_t \setminus \text{eqc}_t(S)$. Moreover, for a set of equivalence classes $\mathcal{Q} \subseteq V_t/\sim_t$, we let $V(\mathcal{Q}) := \bigcup_{Q \in \mathcal{Q}} Q$.

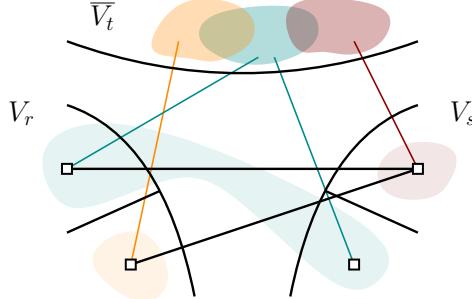


Figure 3.5: Illustration of the operator of a node t in a rooted branch decomposition. The sets V_r and V_s are partitioned into equivalence classes of \sim_r and \sim_s as shown. Each of these equivalence classes corresponds to a vertex of the bipartite graph H_t , illustrated by the square vertices and the edges between them. Furthermore, each equivalence class of \sim_r and \sim_s has a set of neighbors in \bar{V}_t which determines to which equivalence class of \sim_t it belongs. These ‘bubbles’ are illustrated by the lightly shaded regions in $V_r \cup V_s$.

Maximum Independent Set

We now consider the MAXIMUM INDEPENDENT SET problem parameterized by module-width, concretely, we give an FPT-algorithm for the following parameterized version of MAXIMUM INDEPENDENT SET.

MAXIMUM INDEPENDENT SET parameterized by $w := \text{mw}(T, \mathcal{L})$

Input: Graph G and one of its rooted branch decompositions (T, \mathcal{L}) .
Question: What is the size of a maximum independent set in G ?

We do bottom-up dynamic programming along the rooted tree T . At each node $t \in V(T)$, it suffices to store for each subset of equivalence classes $\mathcal{Q} \subseteq V_t/\sim_t$, the largest size of an independent set contained in $V(\mathcal{Q})$. For if we have two independent sets X and Y in G_t that are contained in the same equivalence classes of \sim_t , then for each subset Z of \bar{V}_t , $X \cup Z$ is an independent set in G if and only if $Y \cup Z$ is. Therefore, it suffices to remember the larger set among X and Y .

Lemma 3.15. *Let G be a graph and (T, \mathcal{L}) one of its rooted branch decompositions, and let $t \in V(T)$. Let $X, Y \subseteq V_t$ be two independent sets in G_t such that $\text{eqc}_t(X) = \text{eqc}_t(Y)$. Then, for any set $Z \subseteq \bar{V}_t$, $X \cup Z$ is an independent set in G if and only if $Y \cup Z$ is an independent set in G .*

Proof. Suppose for a contradiction that $X \cup Z$ is an independent set but $Y \cup Z$ is not. Then, there is some edge $yz \in E(G)$ with $y \in Y$ and $z \in Z$. Let $Q \in V_t/\sim_t$ be the equivalence class of \sim_t containing y . By the definition of \sim_t , we have that all vertices in Q are adjacent to z . Since $\text{eqc}_t(X) = \text{eqc}_t(Y)$, X contains some vertex

from Q , say x . Then, $xz \in E(G)$ a contradiction with $X \cup Z$ being an independent set. The other direction follows from the same argument by renaming. \square

This leads to the following definition of the table entries.

Definition of the table entries. Let $t \in V(T)$ be a node of T . For each $\mathcal{Q} \subseteq V_t/\sim_t$, we store

$$\text{tab}[t, \mathcal{Q}] = \max_{X \subseteq V_t} \{|X| : X \text{ is an independent set in } G_t \text{ and } \text{eqc}_t(X) = \mathcal{Q}\}. \quad \triangleleft$$

Before we proceed with the description of how to compute the table entries, we observe that the information stored in the table entries is sufficient to solve the instance at hand. At the root node $\mathbf{r} \in V(T)$, we have that $G_{\mathbf{r}} = G$, and since $\overline{V_{\mathbf{r}}} = \emptyset$, there is precisely one equivalence class of $\sim_{\mathbf{r}}$, namely $V_{\mathbf{r}} = V(G)$.

Observation 3.16. Let $\mathbf{r} \in V(T)$ be the root of T . Once computed, $\text{tab}[\mathbf{r}, \{V(G)\}]$ holds the size of a maximum independent set in G .

We first show how to compute the table entries at leaves of T (by brute force).

Leaves of T . Let $t \in V(T)$ be a leaf of T , and let v be the vertex such that $\mathcal{L}(v) = t$. Then, $V_t = \{v\}$, and there is one equivalence class of \sim_t , namely $\{v\}$; moreover, there are two subsets of equivalence classes of \sim_t , namely the empty set and $\{\{v\}\}$. Clearly, we have to set $\text{tab}[t, \emptyset] := 0$ and $\text{tab}[t, \{\{v\}\}] := 1$. \triangleleft

Internal nodes of T . Let $t \in V(T) \setminus L(T)$ be an internal node of T with children r and s , and assume that the table entries at r and s have been computed. Let I_r be an independent set in G_r and I_s be an independent set in G_s . Then, clearly, $I_r \cup I_s$ is an independent set in G_t if and only if the operator of t does not introduce any edges between I_r and I_s . In other words, if and only if $\text{eqc}_r(I_r) \cup \text{eqc}_s(I_s)$ is an independent set in H_t . Lemma 3.15 suggests that instead of any such pair I_r, I_s , it suffices to consider, for each pair $\mathcal{Q}_r \in V_r/\sim_r, \mathcal{Q}_s \in V_s/\sim_s$, such that $\mathcal{Q}_r \cup \mathcal{Q}_s$ is independent in H_t , a pair of a maximum independent set in $G[V(\mathcal{Q}_r)]$ and a maximum independent set in $G[V(\mathcal{Q}_s)]$. By assumption, the sizes of such sets have been computed.

Let I_r^*, I_s^* be a pair of such independent sets. To determine the equivalence classes of \sim_t containing $I_r^* \cup I_s^*$, we simply consult the maps η_r and η_s : we have that $\text{eqc}_t(I_r^* \cup I_s^*) = \eta_r(\text{eqc}_r(I_r^*)) \cup \eta_s(\text{eqc}_s(I_s^*))$. We give a formal description of the resulting procedure in Algorithm 1. \triangleleft

Lemma 3.17. *For each $t \in V(T)$ and $\mathcal{Q}_t \subseteq V_t/\sim_t$, the above algorithm computes the table entry $\text{tab}[t, \mathcal{Q}_t]$ correctly.*

Proof. We prove the lemma by induction on the height of t . If t is a leaf node, then it is clear, since we computed the table entries at the leaves by brute force.

```

1 foreach  $\mathcal{Q}_t \subseteq V_t / \sim_t$  do  $\text{tab}[t, \mathcal{Q}_t] \leftarrow 0$ ;
2 foreach  $\mathcal{Q}_r \subseteq V_r / \sim_r$ ,  $\mathcal{Q}_s \subseteq V_s / \sim_s$  do
3   if  $\mathcal{Q}_r \cup \mathcal{Q}_s$  is an independent set in  $H_t$  then
4     Let  $\mathcal{Q}_t \leftarrow \eta_r(\mathcal{Q}_r) \cup \eta_s(\mathcal{Q}_s)$ ;
5      $\text{tab}[t, \mathcal{Q}_t] \leftarrow \max\{\text{tab}[t, \mathcal{Q}_t], \text{tab}[r, \mathcal{Q}_r] + \text{tab}[s, \mathcal{Q}_s]\}$ ;

```

Algorithm 1: Algorithm to compute the table entries at internal nodes of the algorithm for MAXIMUM INDEPENDENT SET on graphs of bounded module-width.

Now suppose that t is an internal node with children r and s . We first show that if $\text{tab}[t, \mathcal{Q}_t] = k$, then there is an independent set I_t in G_t of size k such that $\text{eqc}_t(I_t) = \mathcal{Q}_t$. We may assume that $k > 0$. In this case, following Algorithm 1, there are $\mathcal{Q}_r \subseteq V_r / \sim_r$ and $\mathcal{Q}_s \subseteq V_s / \sim_s$ such that

- $\mathcal{Q}_r \cup \mathcal{Q}_s$ is an independent set in H_t ,
- $\mathcal{Q}_t = \eta_r(\mathcal{Q}_r) \cup \eta_s(\mathcal{Q}_s)$, and
- $k = \text{tab}[r, \mathcal{Q}_r] + \text{tab}[s, \mathcal{Q}_s]$.

By induction, the table entries at r and s are computed correctly, so there are independent sets I_r and I_s of G_r and G_s , respectively, such that $\text{eqc}_r(I_r) = \mathcal{Q}_r$ and $\text{eqc}_s(I_s) = \mathcal{Q}_s$, moreover, $k = |I_r| + |I_s|$. By the first item in the list above, there are no edges between \mathcal{Q}_r and \mathcal{Q}_s in H_t , which implies that there are no edges between I_r and I_s . Therefore, $I_t := I_r \cup I_s$ is an independent set of size k in G_t . The second item in the above list implies that $\text{eqc}_t(I_t) = \mathcal{Q}_t$.

For the other direction, suppose that I_t is an independent set in G_t with $\text{eqc}_t(I_t) = \mathcal{Q}_t$. We show that in this case, $\text{tab}[t, \mathcal{Q}_t] \geq |I_t|$. Let $I_r := I_t \cap V_r$ and $I_s := I_t \cap V_s$, $\mathcal{Q}_r := \text{eqc}_r(I_r)$ and $\mathcal{Q}_s := \text{eqc}_s(I_s)$. Since I_t is an independent set, $\mathcal{Q}_r \cup \mathcal{Q}_s$ is an independent set in H_t , and it is clear that $\mathcal{Q}_t = \eta_r(\mathcal{Q}_r) \cup \eta_s(\mathcal{Q}_s)$. By induction, we have that $\text{tab}[r, \mathcal{Q}_r] \geq |I_r|$ and $\text{tab}[s, \mathcal{Q}_s] \geq |I_s|$, so we can conclude that $\text{tab}[t, \mathcal{Q}_t] \geq \text{tab}[r, \mathcal{Q}_r] + \text{tab}[s, \mathcal{Q}_s] \geq |I_r| + |I_s| = |I_t|$. \square

By Lemma 3.17, the algorithm is correct, and by Observation 3.16, the answer to the given instance can be read off the table entries at the root node. It remains to analyze the runtime of the algorithm. Let $w := \text{mw}(T, \mathcal{L})$. At leaf nodes, we clearly spend only constant time. At internal nodes, we try all pairs of at most 2^w subsets of equivalence classes, one per child, and for each such pair, we spend at most $\mathcal{O}(w^2)$ time. Since $|V(T)| = \mathcal{O}(n)$, we have the following theorem.

Theorem 3.18. *There is an algorithm that given a graph G together with one of its rooted branch decompositions (T, \mathcal{L}) of module-width $w := \text{mw}(T, \mathcal{L})$, solves MAXIMUM INDEPENDENT SET on G in time $\mathcal{O}(4^w \cdot w^2 \cdot n)$.*

3.3.1 Comparison with Treewidth

Already in the work introducing clique-width, Courcelle and Olariu [89] showed that graphs of bounded treewidth have bounded clique-width. Concretely, they showed that any graph of treewidth at most w has clique-width at most $4 \cdot 2^{w-1} + 1$. This bound was improved by Corneil and Rotics [83] to $3 \cdot 2^{w-1}$. Interestingly, Corneil and Rotics also ruled out bounds on the clique-width that are subexponential in the treewidth.

Theorem 3.19 (Corneil and Rotics [83]). *For every graph G with $\text{tw}(G) \leq w$, it holds that $\text{cw}(G) \leq 3 \cdot 2^{w-1}$. Moreover, for any positive integer w , there is a graph G with $\text{tw}(G) = w$ and $\text{cw}(G) \geq 2^{\lfloor \frac{w}{2} \rfloor - 1}$.*

On the other hand, it is easy to observe that there are graphs of unbounded treewidth whose clique-width is constant. Take for instance K_n , the complete graph on n vertices. Since in a tree decomposition, for each clique X , there has to be a single bag containing all vertices of X , it immediately follows that the treewidth of K_n is $n-1$. On the other hand, the following 2-expression evaluates to K_n , therefore⁵ K_n has clique-width 2:

$$\underbrace{\rho_{2 \rightarrow 1}(\eta_{1,2}(\bullet_2 \oplus \cdots)}_{n-2 \text{ times}} \rho_{2 \rightarrow 1}(\eta_{1,2}(\bullet_2 \oplus \bullet_1)) \cdots)$$

Interestingly, on *sparse*⁶ classes of graphs, treewidth and clique-width are equivalent. Gurski and Wanke [150] showed that if a graph of bounded clique-width has no biclique of some fixed size, then its treewidth is bounded as well. Biclique-free graphs generalize nowhere dense graphs, the largest class considered under the umbrella term of sparse classes. The following result also shows the equivalence of treewidth and clique-width on bounded degree graphs, since bounded degree graphs are nowhere dense.

Theorem 3.20 (Gurski and Wanke [150]). *Let $t \in \mathbb{N}$. For any $K_{t,t}$ -free graph G of clique-width w , it holds that $\text{tw}(G) \leq 3w(t-1) - 1$.*

3.3.2 Computing Clique-Width

We now discuss the problem of computing clique-width expressions of graphs of bounded clique-width which – as in the case of treewidth – is not so straightforward, and inspired algorithmic and structural inventions. We deal with the following computational problem.

⁵Observe that only edgeless graphs have clique-width 1.

⁶Read: ‘graphs with few edges.’ For an introduction to the topic of sparse graphs we refer to the book by Nešetřil and Ossona de Mendez [230].

CLIQUE-WIDTH EXPRESSION

Input: Graph G , integer k
Task: Construct a clique-width k -expression of G , if it exists; report $\text{cw}(G) > k$, otherwise

The majority of algorithmic as well as complexity results regarding the CLIQUE-WIDTH EXPRESSION problem took some time to be found. At the time Fellows et al. [120] showed that CLIQUE-WIDTH EXPRESSION is NP-complete, the clique-width parameter had already received much attention.

Early on, Corneil et al. [80] showed that graphs of clique-width at most 3 are recognizable in polynomial time; specifically, given an n -vertex m -edge graph, their algorithm runs in time $\mathcal{O}(n^2m)$ and constructs a 3-expression of the input graph if it exists, or reports that its clique-width is greater than 3, otherwise. Ever since, it has been an open problem whether graphs of clique-width $k \geq 4$ can be recognized in polynomial time.

To approximate CLIQUE-WIDTH EXPRESSION, Oum and Seymour [240] introduced a new width measure: the *rank-width* of a graph G , denoted by $\text{rw}(G)$, is defined in terms of branch decompositions over $V(G)$. The rank-width of a branch decomposition is the maximum over all cuts of the $GF(2)$ -rank of the bipartite adjacency matrix induced by the cut. The rank-width of G is the minimum rank-width of all its branch decompositions. Oum and Seymour [240] proved that for each graph G ,

$$\text{rw}(G) \leq \text{cw}(G) \leq 2^{\text{rw}(G)+1} - 1.$$

Moreover, the proof of this inequality is constructive, i.e. given a branch decomposition of (fixed) rank-width k , it outputs in time $\mathcal{O}(n^2)$ a clique-width $(2^{k+1} - 1)$ -expression of G . Combined with an algorithm that given a graph G , either concludes that $\text{rw}(G) > k$, or outputs a branch decomposition of rank-width at most $3k + 1$ in time $\mathcal{O}(8^k \cdot n^9 \log n)$, this led to an algorithm that either concludes that $\text{cw}(G) > k$ or constructs a $(2^{3k+2} - 1)$ -expression of G in the same runtime bound. Oum [236] improved the running time of this approximation to $\mathcal{O}(8^k \cdot n^4)$, and Hliněný and Oum [160] gave an algorithm that computes a branch decomposition of optimum width in time $\mathcal{O}(g(k) \cdot n^3)$, where g is some (fast growing) computable function.⁷ Therefore, for a graph of fixed clique-width k , we can compute a $2^{k+1} - 1$ -expression in time $\mathcal{O}(n^3)$.

Until now, no better approximation ratio for CLIQUE-WIDTH EXPRESSION is known. Therefore, algorithms that rely on clique-width expressions typically suffer from an exponential blow-up in the runtime. Concretely, if a problem can be solved in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ given a k -expression, then using the approximation algorithm

⁷A non-constructive FPT time recognition algorithm for rank-width k was given earlier by Courcelle and Oum [90].

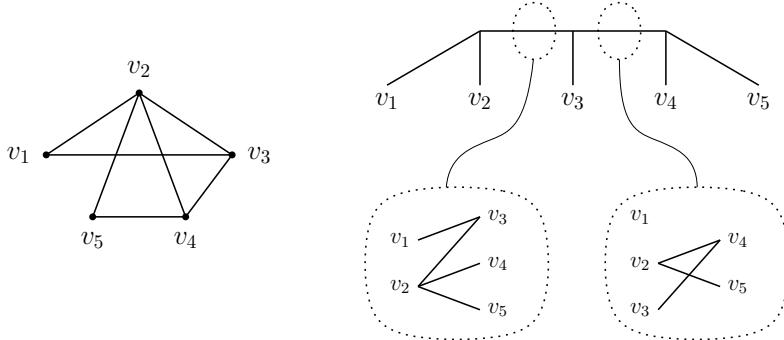


Figure 3.6: A graph on the left and on the right one of its linear branch decompositions of mim-width 1. Each of the cuts induce a bipartite in which there is no induced matching of size 2. (Note that a branch decomposition of this graph having a cut with vertices $\{v_1, v_5\}$ on one side and $\{v_3, v_4\}$ on the other has mim-width 2.)

described above, this only gives a $2^{\mathcal{O}(2^k)} \cdot n^{\mathcal{O}(1)}$ -time algorithm on graphs of clique-width k . By giving algorithms that work on branch decompositions of bounded rank-width instead, several works circumvent this issue, see for instance [60, 239].

It is interesting to note that besides algorithmic applications, the study of rank-width established a deep structure theory, especially through the connection to *vertex minors* [235]. We refer the reader to the survey [238].

3.4 Mim-Width

Maximum induced matching width (mim-width for short) was introduced by Vatshelle [281], and is the most expressive width measure that we consider in this thesis in depth. While bounded clique-width implies bounded mim-width, there are graphs of mim-width 1 whose clique-width is the square-root of the number of vertices.

Mim-width is based on branch decompositions, and it bounds the maximum size of any induced matching in the bipartite subgraphs induced by the cuts in a branch decomposition. Recall that an induced matching M of a graph G is a collection of pairwise disjoint edges, such that there are no other edges between the endpoints of M in G (i.e. $E(G[V(M)]) = M$.) Formally, it is defined as follows (recall Definition 3.6); we give an illustration of a branch decomposition of small mim-width in Figure 3.6.

Definition 3.21 (Mim-Width). Let G be a graph and $\text{mim}_G: 2^{V(G)} \rightarrow \mathbb{N}$ be the set function, where for all $A \subseteq V(G)$,

$$\text{mim}_G(A) := \max\{|M| : M \text{ is an induced matching in } G[A, V(G) \setminus A]\}.$$

The *mim-width* of G is the mim_G -width of G .

We observe that the set function mim_G as in the previous definition is symmetric, therefore we may consider mim-width both in terms of rooted and unrooted branch decompositions. In algorithmic applications, it is typically more convenient to consider the rooted variant, while several structural proofs are simpler when considering the unrooted variant.

Maximum Independent Set

To give the reader an idea of how mim-width based algorithms work, we now give the example of the MAXIMUM INDEPENDENT SET problem. Formally, we consider the following problem.

MAXIMUM INDEPENDENT SET parameterized by $w := \text{mimw}(T, \mathcal{L})$

Input: A graph G and one of its rooted branch decompositions (T, \mathcal{L}) .

Question: What is the size of a maximum independent set in G ?

The exposition of the algorithm presented here is based on the works of Bui-Xuan, Telle, and Vatshelle [61, 62]. The algorithms we presented in the previous sections directly made use of some structural properties of the given decomposition. In the case of the mim-width based algorithm for MAXIMUM INDEPENDENT SET, this happens in a slightly indirect way. Nevertheless, we can draw an analogy with module-width. Algorithms based on rooted branch decompositions of small module-width exploit the property that at each node t , we can partition V_t into a small number of equivalence classes, based on the intersection of the neighborhoods of vertices in V_t with $\overline{V_t}$. The mim-width based algorithm presented in this section considers an equivalence relation of all subsets of V_t , and partitions the power set 2^{V_t} into equivalence classes, again based on the intersection of neighborhoods with $\overline{V_t}$. Formally, we consider the *neighborhood equivalence relation* defined as follows.

Definition 3.22 (Neighborhood Equivalence). Let (T, \mathcal{L}) be a rooted branch decomposition and $t \in V(T)$. For two subsets $X, Y \subseteq V_t$, we say that X and Y are *neighborhood equivalent*, in symbols $X \equiv_t Y$, if $N(X) \cap \overline{V_t} = N(Y) \cap \overline{V_t}$.

The key aspects of the neighborhood equivalence with respect to the problem at hand are:

- (A) For each equivalence class Q_t of \equiv_t , it suffices to remember the largest independent set of G_t that is contained in Q_t .
- (B) The number of equivalence classes of each \equiv_t is bounded by $n^{\text{mim}(V_t)}$.

We first argue item A, and we delay the proof of item B to the end of this section. We prove below that if there are two independent sets X and Y with $X \equiv_t Y$, then for any set $Z \subseteq \overline{V_t}$ such that $X \cup Z$ is independent in G , it holds that $Y \cup Z$ is

independent in G (and vice versa). Now, if $|X| \geq |Y|$, then from any independent set S in G such that $S \cap V_t = Y$, we can obtain an independent set S' that is at least as large as S by replacing Y with X , i.e. we let $S' := (S \setminus Y) \cup X$. Since $|X| \geq |Y|$, it immediately follows that $|S'| \geq |S|$. Therefore we can ‘forget about Y ’ when doing dynamic programming along T . We now prove the above mentioned claim, in a slightly more general form.

Lemma 3.23. *Let (T, \mathcal{L}) be a rooted branch decomposition and $t \in V(T)$, and let $X, Y \subseteq V_t$ be such that $X \equiv_t Y$. Then, for each $Z \in \overline{V}_t$, there are no edges between X and Z if and only if there are no edges between Y and Z .*

Proof. Suppose there are no edges between X and Z , and for the sake of a contradiction that there is some edge $yz \in E(G)$ with $y \in Y$ and $z \in Z$. In other words, $z \in (N(Y) \setminus N(X)) \cap \overline{V}_t$, and therefore $N(X) \cap \overline{V}_t \neq N(Y) \cap \overline{V}_t$, contradicting the fact that $X \equiv_t Y$. \square

This motivates the following definition of the table entries: for each $t \in V(T)$ and each $Q_t \in 2^{V_t}/\equiv_t$, store the largest independent set of G_t contained in Q_t . However, there is one more issue we have to deal with first, namely how to represent the equivalence classes of \equiv_t . An equivalence class may contain up to exponentially (in n) many subsets of V_t , which would lead to table indices of prohibitively large size.

To overcome this issue, we consider the following compact representation of equivalence classes. For any $t \in V(T)$ and any $Q_t \in 2^{V_t}/\equiv_t$, the *description* of Q_t , denoted throughout by $\text{desc}_t(Q_t)$, or simply $\text{desc}(Q_t)$ if t is clear from the context, is the set of neighbors its members have in \overline{V}_t . Formally,

$$\text{desc}(Q_t) := \{S \subseteq \overline{V}_t \mid \forall X \in Q_t: N(X) \cap \overline{V}_t = S\}.$$

Definition of the table entries. For each $t \in V(T)$, and each $Q_t \in 2^{V_t}/\equiv_t$, we let

$$\text{tab}[t, \text{desc}(Q_t)] = \operatorname{argmax}_{S \in Q_t} \{|S| : S \text{ is an independent set in } G_t\}. \quad \square$$

Observe that technically speaking, the index of a table entry is a pair of a node $t \in V(T)$ and some subset of \overline{V}_t . However, we cannot afford to address the table entry for *every* such pair, since there may be exponentially many subset of \overline{V}_t . We therefore assume that whenever a table entry has not been filled yet, it is trivially set to \emptyset . Moreover, once we have computed all table entries at a node t , we can efficiently enumerate all (descriptions of) the equivalence classes of \equiv_t ; we simply have to enumerate all table entries at t that have been filled.

Now, let us observe that the information stored at the table entries is indeed sufficient to solve MAXIMUM INDEPENDENT SET, i.e. that once all table entries have been computed, we can report the correct answer to the given instance. Let $\mathbf{r} \in V(T)$ be the root node of T . Then, $V_{\mathbf{r}} = V(G)$, and since $\overline{V}_{\mathbf{r}} = \emptyset$, there is only one

equivalence class of \equiv_{τ} , namely $2^{V(G)}$, the set of all subsets of $V(G)$. Therefore, the maximum independent set in the equivalence class $2^{V(G)}$ is clearly the maximum independent set of G . Note that $\text{desc}_{\tau}(2^{V(G)}) = \emptyset$.

Observation 3.24. Let G be a graph and (T, \mathcal{L}) be one of its branch decomposition, rooted at a node $\tau \in V(T)$, and let $\text{tab}[\cdot, \cdot]$ be the table defined above. If all table entries have been computed correctly, then $\text{tab}[\tau, \emptyset]$ contains a maximum independent set of G .

We now show how to compute the table entries, starting at the leaves of T . For simplicity we assume that the input graph G is connected; note that if G is not connected we can simply solve the problem on each of its connected components separately, and put together the solutions.

Leaves of T . Let $t \in V(T)$ be a leaf of T . Then, $V_t = \{v\}$ for some vertex $v \in V(G)$, and there are two equivalence classes of \equiv_t , namely $\{\emptyset\}$ and $\{\{v\}\}$. (Since G is connected, v has at least one neighbor in $V(G) \setminus \{v\}$ and therefore \emptyset and $\{v\}$ are in different equivalence classes of \equiv_t .) Moreover, $\text{desc}(\{\emptyset\}) = \emptyset$ and $\text{desc}(\{\{v\}\}) = N(v)$. We set $\text{tab}[t, \text{desc}(\{\emptyset\})] = 0$, since there is no non-empty set in the equivalence class $\{\emptyset\}$, and $\text{tab}[t, \text{desc}(\{\{v\}\})] = 1$, since $\{v\}$ is the only (and therefore largest, and trivially independent) set in the equivalence class $\{\{v\}\}$. \triangleleft

Before we describe the algorithm to fill the table entries at internal nodes, we discuss one more important property of the neighborhood equivalence relation, namely the equivalence relations *coarsen* as we move up the tree T .

Concretely, let p be an ancestor of t . If for two subsets $X, Y \subseteq V_t$, we have that $X \equiv_t Y$, then $N(X) \cap \overline{V_t} = N(Y) \cap \overline{V_t}$. Since $V_t \subseteq V_p$, we have that $\overline{V_p} \subseteq \overline{V_t}$, and therefore $N(X) \cap \overline{V_p} = N(Y) \cap \overline{V_p}$, in other words, $X \equiv_p Y$.

Observation 3.25. Let (T, \mathcal{L}) be a rooted branch decomposition, let $t \in V(T)$ and let $p \in V(T)$ be an ancestor of t . For all $X, Y \subseteq V_t$, if $X \equiv_t Y$ then $X \equiv_p Y$.

Now suppose that t is an internal node and a and b are its children. Observation 3.25 implies that to obtain the equivalence classes of \equiv_t , we do not have to ‘split up’ equivalence classes of \equiv_a or \equiv_b . Therefore, knowing the equivalence classes of \equiv_a and \equiv_b , we can enumerate all equivalence classes of \equiv_t by considering at most $|V_a / \equiv_a| \cdot |V_b / \equiv_b|$ subsets of $\overline{V_t}$ as descriptions of equivalence classes of \equiv_t .

Internal nodes of T . Let $t \in V(T) \setminus L(T)$ be an internal node with children a and b . Let I_a be an independent set in G_a and I_b an independent set in G_b . Then, $I_t := I_a \cup I_b$ is an independent set in G_t if and only if there are no edges between I_a and I_b . Moreover, we can determine directly which equivalence class of \equiv_t the set I_t belongs by computing its description: since $I_t = I_a \cup I_b$, we have that $N(I_t) = N(I_a) \cup N(I_b)$, and therefore the description of the equivalence class of \equiv_t containing I_t is $(N(I_a) \cup N(I_b)) \cap \overline{V_t}$. In our algorithm, we consider I_a and I_b only as members

```

1 foreach  $Q_a \in 2^{V_a}/\equiv_a$ ,  $Q_b \in 2^{V_b}/\equiv_b$  do
2   Let  $X_a \leftarrow \text{tab}[a, \text{desc}(Q_a)]$  and  $X_b \leftarrow \text{tab}[b, \text{desc}(Q_b)]$ ;
3   if there are no edges between  $X_a$  and  $X_b$  then
4     Let  $S := (\text{desc}(Q_a) \cup \text{desc}(Q_b)) \cap \overline{V_t}$ ;
5     if  $|X_a \cup X_b| \geq |\text{tab}[t, S]|$  then
6        $\text{tab}[t, S] \leftarrow X_a \cup X_b$ ;

```

Algorithm 2: Algorithm to compute the table entries at internal nodes of the algorithm for MAXIMUM INDEPENDENT SET on graphs of bounded mim-width. Note that we can enumerate the equivalence classes of \equiv_a and \equiv_b by considering the table entries that have been filled at nodes a and b , respectively.

of some equivalence classes, and it may seem that verifying whether or not there are edges between I_a and I_b depends on the concrete choice of such sets. However, using Lemma 3.23 we prove below that knowing which equivalence classes I_a and I_b belong to suffices to verify this condition as well.

For each pair $Q_a \in 2^{V_a}/\equiv_a$, $Q_b \in 2^{V_b}/\equiv_b$, we consider a pair of a *largest* independent set in Q_a and one in Q_b when combining them to potentially maximum independent sets in G_t . By Observation 3.25, we know that each equivalence class of \equiv_t is the result of ‘merging’ some pairs of an equivalence class of \equiv_a and an equivalence class of \equiv_b . We therefore arrive at the procedure described in Algorithm 2 to compute the table entries at t . \triangleleft

We now show that the above algorithm is correct.

Lemma 3.26. *For each $t \in V(T)$, the above algorithm computes the table entries $\text{tab}[t, \cdot]$ correctly.*

Proof. We prove the lemma by induction on the height of t . In the base case, when t is a leaf, correctness is immediate since we computed the table entries at the leaves by brute force.

Now suppose that t is an internal node with children a and b . Let $Q_t \in 2^{V_t}/\equiv_t$. We first show that $X_t = \text{tab}[t, \text{desc}(Q_t)]$ is an independent set of G_t contained in Q_t . Since the algorithm entered X_t at the entry $\text{tab}[t, \text{desc}(Q_t)]$, we know that there are sets $X_a \subseteq V_a$ and $X_b \subseteq V_b$ which are contained in some equivalence class Q_a of \equiv_a and Q_b of \equiv_b , respectively, such that

- $X_t = X_a \cup X_b$,
- there are no edges between X_a and X_b , and
- X_a was stored at table entry $\text{tab}[a, \text{desc}(Q_a)]$ and X_b was stored at table entry $\text{tab}[b, \text{desc}(Q_b)]$.

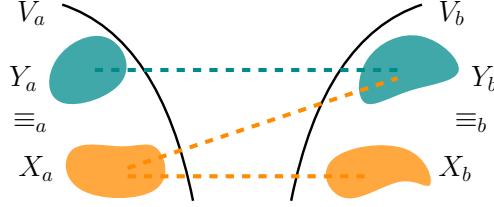


Figure 3.7: Visualization of an argument in the proof of Lemma 3.26. By choice, there are no edges between Y_a and Y_b , which allows us to conclude that there are no edges between X_a and X_b using the neighborhood equivalence.

By the first and second item, if X_t is not an independent set, then either X_a or X_b is not an independent set. However, by the third item and induction, both X_a and X_b are independent sets.

To see that $\text{tab}[t, Q_t]$ stores a *maximum* independent set, let Y_t be an independent set in G_t such that $Y_t \in Q_t$. We show that after executing Algorithm 2, $\text{tab}[t, \text{desc}(Q_t)]$ holds an independent set $Z \in Q_t$ with $|Z| \geq |Y_t|$. Let $Y_a := Y_t \cap V_a$ and $Y_b := Y_t \cap V_b$, and denote by Q_a the equivalence class of \equiv_a containing Y_a and by Q_b the equivalence class of \equiv_b containing Y_b . By induction, we have that X_a and X_b as obtained in line 2 are independent sets in Q_a and Q_b , respectively, with $|X_a| \geq |Y_a|$ and $|X_b| \geq |Y_b|$, so $|X_a \cup X_b| \geq |Y_a \cup Y_b| = |Y_t|$. Since Y_t is an independent set in G_t , there are no edges between Y_a and Y_b .

With help of Lemma 3.23, we can conclude that there are no edges between X_a and X_b , see Figure 3.7. Since $X_a \equiv_a Y_a$ and there are no edges between Y_a and Y_b , we have by Lemma 3.23 that there are no edges between X_a and Y_b . Since $X_b \equiv_b Y_b$, and there are no edges between X_a and Y_b , there are no edges between X_a and X_b , again by Lemma 3.23.

It remains to show that $\text{desc}(Q_t) = (\text{desc}(Q_a) \cup \text{desc}(Q_b)) \cap \overline{V_t}$. If this holds, then the algorithm updated $\text{tab}[t, \text{desc}(Q_t)]$ with $X_a \cup X_b$, see lines 4 to 6. We have that

$$\begin{aligned} \text{desc}(Q_t) &= N(X_t) \cap \overline{V_t} = N(X_a \cup X_b) \cap \overline{V_t} \\ &= ((N(X_a) \cap \overline{V_a}) \cup (N(X_b) \cap \overline{V_b})) \cap \overline{V_t} \\ &= (\text{desc}(Q_a) \cup \text{desc}(Q_b)) \cap \overline{V_t}. \end{aligned}$$

(For the second step, note that for all $c \in \{a, b\}$, $\overline{V_c} \supseteq \overline{V_t}$.) □

By Lemma 3.26 we know that the table entries are computed correctly by the above described algorithm, and by Observation 3.24, we know that the solution to the instance at hand can be read off the table entries at the root, once computed. It remains to argue the runtime. To that end, let⁸ $\text{nec}(T, \mathcal{L}) := \max_{t \in V(T)} |2^{V_t}| / |\equiv_t|$. We

⁸Read: ‘(maximum) number of equivalence classes’.

first give the runtime bound in terms of $\text{nec}(T, \mathcal{L})$, and then show that this quantity is upper bounded in terms of $n^{\text{mimw}(T, \mathcal{L})}$.

At leaf nodes, it is clear that the computation takes constant time. At internal nodes, there are at most $\text{nec}(T, \mathcal{L})^2$ pairs of equivalence classes to consider, and for each such pair, the remaining operations take no more than $\mathcal{O}(n^2)$ time. Since there are $\mathcal{O}(n)$ nodes in $V(T)$, the total runtime is $\mathcal{O}(\text{nec}(T, \mathcal{L})^2 \cdot n^3)$.

Theorem 3.27 (Bui-Xuan et al. [61, 62]). *There is an algorithm that given a graph G and one of its rooted branch decompositions (T, \mathcal{L}) , computes a maximum size independent set of G in time $\mathcal{O}(\text{nec}(T, \mathcal{L})^2 \cdot n^3)$*

To obtain an XP-runtime in the targeted parameterization by the *mim-width* of the given branch decomposition, we now state and prove the following bound on the number of equivalence classes in \equiv_t due to Belmonte and Vatshelle [17].

Lemma 3.28 (Belmonte and Vatshelle [17]). *Let G be a graph on n vertices and let (T, \mathcal{L}) be one of its rooted branch decompositions. For every $t \in V(T)$, $|2^{V_t}/\equiv_t| \leq n^{\text{mim}(V_t)}$.*

Proof. We prove the lemma through the following claim, which essentially states that if the size of a maximum induced matching over the cut (V_t, \bar{V}_t) is at most k , then each equivalence class of \equiv_t contains a set of size at most k . Therefore, we know that when we consider all subsets of V_t of size at most k , we will have seen at least one element of each equivalence class of \equiv_t , and the claimed bound follows.

Claim. Let G and t be as above. Then, $\text{mim}(V_t) \leq k$ if and only if for every $S \subseteq V_t$, there is a set $R \subseteq S$ such that $|R| \leq k$ and $R \equiv_t S$.

Proof. We prove the claim via its contrapositive: $\text{mim}(V_t) > k$ if and only if there is some $S \subseteq V_t$ such that for every $R \subseteq S$, either $|R| > k$ or $N(R) \cap \bar{V}_t \neq N(S) \cap \bar{V}_t$.

For the first direction, suppose that $\text{mim}(V_t) > k$. Then there is some induced matching M in $G[V_t, \bar{V}_t]$ of size more than k . Let $S := V(M) \cap V_t$. Now, consider any $R \subseteq S$. If $R = S$, then $|R| = |S| > k$, so one of the targeted conditions is satisfied. If $R \subset S$, then there is some vertex $u \in S \setminus R$. Let $v \in \bar{V}_t$ be the vertex that u is matched to in M , i.e. such that $uv \in M$. Since M is induced, we know that v is not adjacent to any vertex in S but to u . Therefore, v is a vertex of $(N(S) \cap \bar{V}_t) \setminus (N(R) \cap \bar{V}_t)$, and therefore $N(S) \cap \bar{V}_t \neq N(R) \cap \bar{V}_t$.

For the other direction, let $S \subseteq V_t$ be an inclusion-wise minimal set such that for every $R \subseteq S$, either $|R| > k$ or $N(R) \cap \bar{V}_t \neq N(S) \cap \bar{V}_t$. We construct an induced matching of size more than k in $G[V_t, \bar{V}_t]$. First, we observe that $|S| > k$; for if not, then we immediately obtain a contradiction with our assumption on S (take $R = S$). Next, if there is some $v \in S$ such that $N(S \setminus \{v\}) \cap \bar{V}_t = N(S) \cap \bar{V}_t$, then we obtain a contradiction with minimality of S . For every $v \in S$, we trivially have that $N(S \setminus \{v\}) \cap \bar{V}_t \subseteq N(S) \cap \bar{V}_t$, so by the previous argument, $N(S \setminus \{v\}) \cap \bar{V}_t \subset N(S) \cap \bar{V}_t$, in other words there is a vertex $\bar{v} \in \bar{V}_t$ that is adjacent to v and non-adjacent to all

other vertices in S . Then, $M := \{v\bar{v} \mid v \in S\}$ is an induced matching in $G[V_t, \bar{V}_t]$, and $|M| = |S| > k$. \square

The previous claim asserts that each equivalence class of \equiv_t has an element whose size is at most the size of a maximum induced matching of $G[V_t, \bar{V}_t]$. Therefore, \equiv_t has most $|V_t|^{\text{mim}(V_t)} \leq n^{\text{mim}(V_t)}$ equivalence classes. \square

We obtain the algorithm we aimed for in this section by combining Theorem 3.27 with Lemma 3.28.

Corollary 3.29 (Belmonte and Vatshelle [17], Bui-Xuan et al. [61, 62]). *There is an algorithm that given a graph G together with one of its branch decompositions (T, \mathcal{L}) of mim-width $w := \text{mimw}(T, \mathcal{L})$, solves MAXIMUM INDEPENDENT SET on G in time $\mathcal{O}(n^{2w+3})$.*

The avid reader may have noticed that the arguments that led to the mim-width based algorithm are very similar to the ones given in the previous section on module-width. The reason is that the neighborhood equivalence relation generalizes the ‘module-width equivalence relation’, as we see now. (For the following lemma, recall that \sim_t denotes the equivalence relation of V_t with respect to module-width.)

Lemma 3.30 (Vatshelle [281]). *Let G be a graph on n vertices and let (T, \mathcal{L}) be one of its rooted branch decompositions. For every $t \in V(T)$, $|2^{V_t}/\equiv_t| \leq 2^{|V_t|/\sim_t|}$.*

Proof. To avoid confusion, for a set $X \subseteq V_t$, we denote by $\text{eqc}_{\sim_t}(X)$ the equivalence classes of \sim_t whose intersection with X is nonempty. We show that for each pair of sets $X, Y \subseteq V_t$, if $\text{eqc}_{\sim_t}(X) = \text{eqc}_{\sim_t}(Y)$, then $X \equiv_t Y$, from which the lemma follows.

Suppose $\text{eqc}_{\sim_t}(X) = \text{eqc}_{\sim_t}(Y)$ and let $v \in N(X) \cap \bar{V}_t$. Then, v has a neighbor in some equivalence class $Q \in \text{eqc}_{\sim_t}(X)$. Since $\text{eqc}_{\sim_t}(X) = \text{eqc}_{\sim_t}(Y)$, $Q \in \text{eqc}_{\sim_t}(Y)$, and since all vertices in Q are twins w.r.t. \bar{V}_t , all vertices in Q are adjacent to v . Therefore, there is a vertex in Y that is adjacent to v . This shows that $N(X) \cap \bar{V}_t \subseteq N(Y) \cap \bar{V}_t$ and the other inclusion follows by renaming. \square

Theorem 3.27 combined with Lemma 3.30 therefore implies the following. Note however that the ad-hoc method that led to Theorem 3.18 gave a runtime whose dependence on n was much better, namely linear. Here, the dependence is cubic.

Corollary 3.31. *There is an algorithm that given a graph G together with one of its rooted branch decompositions (T, \mathcal{L}) of module-width $w := \text{mw}(T, \mathcal{L})$, solves MAXIMUM INDEPENDENT SET on G in time $\mathcal{O}(4^w \cdot n^3)$.*

3.4.1 Efficiently Solvable Problems

We briefly review which problems are currently known to be solvable parameterized by the mim-width of a given branch decomposition. Throughout this section, we simply refer to such algorithms as ‘XP-time algorithms parameterized by mim-width’.

Generalizing the techniques used above to solve MAXIMUM INDEPENDENT SET, it follows from results due to Bui-Xuan et al. [62] and Belmonte and Vatshelle [17] that a large number of problems related to independence and domination is XP parameterized by mim-width, the (σ, ρ) -domination and *locally checkable vertex partitioning* (LCVP) problems. In joint work with Kwon, Strømme, and Telle we showed that all *distance* versions in which the independence or domination condition is made within some radius from each vertex rather than the direct neighborhood, admit such algorithms as well. We refer to Section 3.6 for an overview of these problems. Galby et al. [134] gave an XP-algorithm for SEMITOTAL DOMINATING SET parameterized by mim-width, a problem that does not fall under the above mentioned categories. A *semitotal dominating set* of a graph G is a dominating set $D \subseteq V(G)$, such that each vertex of D is at distance at most two to another member of D .

In all of the above problems (think about MAXIMUM INDEPENDENT SET), solutions are sets of vertices, and we can verify if a given vertex set S is a solution simply by independently inspecting, for each vertex v of the graph, the interaction of the vertices at some fixed distance from v with S . The first problems whose solutions *cannot* be verified in such a way that were shown to be XP parameterized by mim-width are problems related to finding induced paths. Specifically, the problems LONGEST INDUCED PATH, INDUCED DISJOINT PATHS, and H -INDUCED TOPOLOGICAL MINOR, for fixed graphs H . An algorithm for FEEDBACK VERTEX SET followed. All these algorithms have been obtained in joint work with Kwon and Telle, and are presented in Chapter 8.

Our algorithms are problem-specific and based on ad-hoc combinatorial lemmas. Bergougnoux and Kanté [21], see also [19], gave a generic framework for problems asking for (σ, ρ) -sets and their complements that are also required to induce a connected and/or acyclic subgraph, which give XP-algorithms parameterized by mim-width as well. This generalizes the above mentioned algorithms. Their algorithms combine the d -neighborhood equivalence relation with ideas from the *rank-based approach* due to Bodlaender et al. [33], which was one of the breakthrough methods to design deterministic single-exponential time algorithms for connectivity problems parameterized by treewidth. In follow-up work [20], Bergougnoux and Kanté also showed that the connectivity and acyclicity requirements can be added to vertex partitioning problems (including LCVP problems, see Section 3.6). This implies that problems such as PARTITION INTO FORESTS which asks if the vertex set of a graph can be partitioned into q sets that induce forests, can be solved in XP time (when also parameterized by q , the number of blocks in the partition). Recently, Bergougnoux et al. [22] used similar methods to give XP-time algorithms for NODE MULTIWAY CUT and SUBSET FEEDBACK VERTEX SET.

All algorithms discussed here run in XP time parameterized by the mim-width of a given branch decomposition of the input graph, so it is natural to wonder whether we can obtain FPT-algorithms. Fomin et al. [127] showed that the INDEPENDENT SET and DOMINATING SET problems are W[1]-hard in this parameterization, and in joint

work with Kwon, Strømme and Telle, we extended their methods to show hardness for several other (σ, ρ) -domination problems, as well as for FEEDBACK VERTEX SET. See Chapter 9.

3.4.2 Comparison with Clique-Width

We now show that mim-width has strictly more expressive power than clique-width. In one direction, we can observe that in any rooted branch decomposition of a graph, the mim-width is no more than its module-width. Let (T, \mathcal{L}) be some rooted branch decomposition of a graph G , and let $t \in V(T)$. Suppose for a contradiction that \sim_t has k equivalence classes, i.e. the number of subsets of \overline{V}_t that form the neighborhood of some vertex in V_t is k , but that $G[V_t, \overline{V}_t]$ contains an induced matching M of size $k+1$. Let $u_1v_1, \dots, u_{k+1}v_{k+1}$ denote the edges of M , such that for all $i \in [k+1]$, $u_i \in V_t$ and $v_i \in \overline{V}_t$. Since for each $i \in [k+1]$, u_i has a neighbor in \overline{V}_t (namely v_i) that is not adjacent to any u_j , $j \neq i$, we have that u_1, \dots, u_{k+1} all must be in distinct equivalence classes of \sim_t . This means that \sim_t has at least $k+1$ equivalence classes, a contradiction.

We have argued that for each graph G , $\text{mimw}(G) \leq \text{mw}(G)$; by Theorem 3.14, $\text{mw}(G) \leq \text{cw}(G)$, and therefore we have:

Theorem 3.32 (Vatshelle [281]). *For each graph G , $\text{mimw}(G) \leq \text{cw}(G)$.*

On the other hand, Golumbic and Rotics [148] showed that there exist interval graphs and permutation graphs on n vertices whose clique-width is at least \sqrt{n} . Belmonte and Vatshelle [17] (see also Section 4.7) showed that all interval and permutation graphs have (linear) mim-width 1, and therefore we have the following theorem.

Theorem 3.33 ([17, 148]). *For infinitely many $n \in \mathbb{N}$, there are n -vertex graphs whose clique-width is at least \sqrt{n} and whose linear mim-width is 1.*

3.4.3 Computing Mim-Width

Based on hardness results about finding induced matchings in bipartite graphs due to Moser and Sikdar [225] and Elbassioni et al. [109], Sæther and Vatshelle [260] showed that computing the mim-width of a graph is NP-hard, W[1]-hard parameterized by the target width value, and that it cannot be constant-factor approximated in polynomial time unless NP = ZPP.

In terms of exact exponential time algorithms, Vatshelle [281] observed that by a combination of results due to Oum [237] and Binkele-Raible et al. [24], there is an $\mathcal{O}^*(2.79^n)$ time algorithm that computes a branch decomposition with smallest mim-width of any n -vertex graph. Specifically, Oum [237] showed that an optimal branch decomposition can be found in time $\mathcal{O}^*(2^n \cdot \text{eval})$ for *any* width measure based on branch decompositions, where eval denotes the time it takes to compute the value of a

cut. Binkele-Raible et al. [24] showed that computing a maximum induced matching in a bipartite graph with n vertices can be done in time $\mathcal{O}^*(1.392^n)$.

All known **XP**-results for problems on graphs of bounded mim-width rely on a previously computed decomposition of small mim-width. In Section 4.7 we will see that many well-studied graph classes have bounded mim-width, and typically, the proofs are constructive. This means that a recognition algorithm for the graph class in question can be utilized to compute branch decompositions of bounded mim-width. If the recognition algorithm for such a class runs in polynomial time, this immediately implies polynomial-time algorithms for a plethora of problems on the graph class at hand.

For the special case of computing the *linear* mim-width of n -vertex trees, Høgemo et al. [161] recently gave an $\mathcal{O}(n \log n)$ -time algorithm. To make full use of **XP**-algorithms parameterized by mim-width, we have to know how to $f(k)$ -approximate mim-width k in **XP** time. We therefore repeat the following open question, which is also stated in e.g. [19, 281].

Open Problem 3.2. Is there some function $f: \mathbb{N} \rightarrow \mathbb{N}$ and an algorithm that given a graph G and an integer k , either decides that $\text{mimw}(G) > k$, or outputs a branch decomposition of G of mim-width at most $f(k)$, and runs in **XP** time parameterized by k ?

We believe that approximating mim-width 1 in polynomial time is an important step towards such an algorithm.

Open Problem 3.3. Is there some constant c and an algorithm that given a graph G , either decides that $\text{mimw}(G) > 1$, or outputs a branch decomposition of G of mim-width at most c , and runs in polynomial time?

Regarding the computation of mim-width 1, we would like to remark that the seemingly related problem that takes as input a graph G and asks whether G has a non-trivial *chain cut* has been shown to be **NP**-complete by Rusu and Spinrad [259]. Chain cuts are precisely the cuts that induce bipartite graphs whose maximum induced matching has size 1, so at first glance it may seem that the hardness carries over. However, requiring that *every* cut in a branch decomposition is a chain cut imposes additional structure to the problem which could be exploited algorithmically.

3.5 Monadic Second Order Logic of Graphs

The amount of literature on **FPT**-algorithms for problems parameterized by treewidth or clique-width is humongous; surveying it goes way beyond the intention of this chapter and the scope of this thesis. However, to give the reader a glimpse into the vast space of problems that are **FPT** parameterized by treewidth (or clique-width), we end this chapter on an introduction to arguably the most famous meta-theorem

in parameterized algorithms: namely that all graph properties that are expressible in monadic second order logic are linear-time FPT parameterized by treewidth.

While the algorithms that come out of this framework may not be practical, this theorem provides a powerful FPT-*classification tool*. For instance, when encountering a new problem parameterized by treewidth (or clique-width), we may be able to quickly determine that it is FPT simply by writing down a few lines of logical symbols. After that, we are confident that there is *some* FPT-algorithm and can therefore focus on working out a practical FPT-algorithm, instead of wasting our time trying to prove hardness results.

Before we give the definition of this logic, let us consider the running example *independent set*. Step by step, we will translate the property that some vertex set X in a graph G is an independent set into monadic second order logic, without knowing for now what that actually is. First, we unravel the definition of X being an independent set (in an intentionally overcomplicated way), and we arrive at:

‘For all pairs of distinct vertices u and v such that u and v are contained in X ,
there is no edge e in G that has u and v as its endpoints.’

MSO logic equips us with some ‘basic tests’, namely: we can verify if two elements are the same or not, whether an element is contained in some set, and whether or not a vertex v and an edge e are incident (written ‘ $\text{inc}(v, e)$ ’).

‘For all pairs of vertices u and v where it does not hold that $u = v$ and such that
 $u \in X$ and $v \in X$, there is no edge e in G such that $\text{inc}(u, e)$ and $\text{inc}(v, e)$.’

The next building block is the use of the *connectives* or (\vee) , and (\wedge) , negation (\neg) , and implication (\Rightarrow) . Applying these in all places we can, we arrive at:

‘For all pairs of vertices u and v , $(\neg(u = v) \wedge u \in X \wedge v \in X) \Rightarrow$ there is no edge e
in G such that $\text{inc}(u, e) \wedge \text{inc}(v, e)$.’

Lastly, we can use existential (\exists) and universal (\forall) quantification over variables, and we arrive at the following MSO sentence:

$$\forall u \forall v ((\neg(u = v) \wedge u \in X \wedge v \in X) \Rightarrow \neg(\exists e (\text{inc}(u, e) \wedge \text{inc}(v, e))))$$

It is common practice to use several shortcuts for frequently used subformulas; for instance ‘ $\forall v(v \in X)$ ’ can be replaced by simply ‘ $\forall v \in X$ ’ or ‘ $\neg(u = v)$ ’ by ‘ $u \neq v$ ’. Cleaning up the previous formula then gives the following:

$$(\forall u \in X)(\forall v \in X)((u \neq v) \Rightarrow \neg(\exists e (\text{inc}(u, e) \wedge \text{inc}(v, e)))) \quad (3.3)$$

Let us get a little more formal. Sentences in MSO logic on graphs are formed by composing *atomic formulas* that involve *variables* that are meant to represent (sets of) vertices or edges of a graph via logical *connectives*. For the time being, we use the notation ‘ V ’ and ‘ E ’ as placeholders for the vertex set and edge set of any graph;

the variables in our logic are then either elements or subsets of V and E . Given a sentence φ in this logic and a graph G , we can *evaluate* φ at G by replacing V with $V(G)$ and E with $E(G)$. Given the properties of the graph G , this evaluation returns either **true** or **false**. If φ evaluated at G is **true**, then we say that G *models* φ and write ' $G \models \varphi$ '. Before we proceed with the exact definition of the logic, we define the following meta-problem.

MSO MODEL-CHECKING

Input: A graph G , a sentence φ in monadic second order logic.
Task: Decide whether or not $G \models \varphi$.

Let us define the logic. We explain all of the building blocks directly in terms of how they behave in an evaluation. As mentioned above, the variables are either elements (called *element variables*) or subsets (called *set variables*) of one of the sets V and E . The above mentioned atomic formulas are:⁹

Equality. For two element variables x and y , ' $x = y$ ' evaluates to **true** if and only if x and y are the same element.

Membership. For an element variable x and a set variable X , ' $x \in X$ ' evaluates to **true** if and only if x is contained in X .

Incidence. For element variables $v \in V$ and $e \in E$, ' $\text{inc}(v, e)$ ' evaluates to **true** if and only if v is incident with e .

We compose these atomic formulas via the following connectives. (Note that implication (\Rightarrow) as used in the above example can be built using ' \neg ' and ' \wedge ').

Or (\vee). For two formulas ψ_1 and ψ_2 , ' $\psi_1 \vee \psi_2$ ' evaluates to **true** if and only if *at least one* of ψ_1 and ψ_2 evaluates to **true**.

And (\wedge). For two formulas ψ_1 and ψ_2 , ' $\psi_1 \wedge \psi_2$ ' evaluates to **true** if and only if *both* of ψ_1 and ψ_2 evaluate to **true**.

Negation (\neg). For a formula ψ , ' $\neg\psi$ ' evaluates to **true** if and only if ψ evaluates to **false**.

Finally, we allow existential (\exists) and universal (\forall) quantification over element variables and set variables. A variable x is called *free* in some formula ψ if it is *not* in the scope of some quantifier. That is, ψ contains neither ' $\exists x$ ' nor ' $\forall x$ '. In that case, we may write ψ as ' $\psi(x)$ '.

⁹To avoid overcomplicating things, we do not distinguish between *variables* and *elements/sets* in graphs at which these formulas are being evaluated in the following list. For instance, when writing ' v is incident with e ', we of course mean that the replacement of the variables v and e by a concrete vertex and edge of some graph are incident.

Existential (\exists). For an element/set variable x that is free in some formula ψ , ' $\exists x(\psi(x))$ ' evaluates to **true** if and only if *there exists* some element/set in the scope of x such that makes ψ evaluate to **true**.

Universal (\forall). For an element/set variable x that is free in some formula ψ , ' $\forall x(\psi(x))$ ' evaluates to **true** if and only if *for all* elements/sets in the scope of x , ψ evaluates to **true**.

Now, an expression that is built in this way is called an *(MSO-) formula*. A formula without free variables is called an *(MSO-) sentence*. We denote this version of MSO as MSO_2 . Later, we consider the restriction MSO_1 , which forbids quantification over edge sets.

We are now ready to state the first main result we wanted to discuss in this section, namely that MSO MODEL-CHECKING is linear FPT time solvable parameterized by the length of the formula and the treewidth of the input graph, which is known as *Courcelle's Theorem* [85]. The algorithms constructed in the proof of Courcelle's Theorem require a tree decomposition of the input graph of bounded width to be provided at the input. Thanks to Bodlaender's Theorem [29], we can find such tree decompositions in linear FPT time.

Theorem 3.34 (Courcelle [85] and Bodlaender [29]). *There is an algorithm, that given a graph G and an MSO_2 -sentence φ , decides whether $G \models \varphi$ in time $f(|\varphi|, \text{tw}) \cdot n$, where n is the number of vertices and tw the treewidth of G .*

Similar results have been obtained independently by Arnborg, Lagergren, and Seese [7], and by Borie, Parker, and Tovey [49]. In Equation (3.3), we have an MSO formula describing that a given set is an independent set. However, this does not immediately imply an FPT-algorithm for MAXIMUM INDEPENDENT SET parameterized by treewidth using the previous theorem. In monadic second order logic, we cannot count the exact size of a set unless it is small; the only MSO formula to verify if a set has size at least k has length $\mathcal{O}(k)$, and if k is unbounded, this is prohibitively long. However, optimization versions of Courcelle's Theorem have been shown as well.

Theorem 3.35 ([7, 29, 49]). *Let $\text{opt} \in \{\max, \min\}$. There is an algorithm, that given a graph G and an MSO_2 -formula φ with one free (set) variable computes the (vertex/edge) set X^* in G which is such that*

$$X^* = \text{arg}\text{opt}_X \{|X| : \varphi(X) \text{ is true}\}$$

in time $f(|\varphi|, \text{tw}) \cdot n$ where n is the number of vertices and tw the treewidth of G .

When trying to extend these results to graphs of bounded clique-width, there is an immediate obstacle. The property that a given edge set forms a *Hamiltonian cycle* of a graph, i.e. a simple cycle that uses every vertex of G , is expressible in MSO_2 .

However, it is known that the HAMILTONIAN CYCLE problem is W[1]-hard parameterized by clique-width [125]. This rules out the possibility of a direct clique-width analogue of the above theorems, as such a theorem would give an FPT-algorithm for HAMILTONIAN CYCLE parameterized by clique-width.

If we restrict ourselves to a slightly weaker form of MSO logic, namely one that forbids quantification over edge sets, then there is good news. Courcelle, Makowsky, and Rotics [88] showed that for the resulting MSO_1 logic, there is an analogue of Courcelle's Theorem in the clique-width parameterization. However, we do not know how to compute clique-width expressions in linear FPT time, therefore the dependence on n is cubic in the following theorem. Once the clique-width expression is computed, the runtime is again linear in n . The following theorem also holds in the optimization variant.

Theorem 3.36 (Courcelle, Makowsky, and Rotics [88]; [160]). *There is an algorithm that given a graph G and an MSO_1 -sentence φ , decides whether $G \models \varphi$ in time $f(|\varphi|, \text{cw}) \cdot n^3$, where n is the number of vertices and cw the clique-width of G .*

For mim-width, we cannot expect such general theorems, not even if we ask for XP-time algorithms, i.e. with running time $n^{f(|\varphi|, \text{mimw})}$. Vatshelle [281] observed that the MAXIMUM CLIQUE problem, asking for the largest subset of pairwise adjacent vertices, is NP-hard on graphs of mim-width at most 6. The property that a set of vertices is a clique can easily be expressed with an MSO_1 -formula. On the other hand, this does not rule out that some more exotic logic still admits such XP-time algorithms (assuming a branch decomposition of bounded mim-width is given), but so far, no such result is known. The most general meta-theorem for mim-width is that for *locally checkable vertex subset and partitioning problems*; we will discuss these problems in Section 3.6.

Open Problem 3.4. Is there some fragment of MSO/First Order Logic of graphs, containing a problem that is not locally checkable, and whose corresponding MODEL CHECKING problem is solvable in time $n^{f(|\varphi|, \text{mimw})}$, if a branch decomposition of mim-width mimw of the input graph is provided?

Complexity Aspects

One may wonder about the exact complexity that the functions $f(|\varphi|, \cdot)$ in the above theorems are hiding, and the answer is: something monstrous. The large degree of generality of these results comes at a price, the function f has a *non-elementary* dependence on $|\varphi|$ in the worst case. Recall that a function g is elementary if there is some positive constant h , such that:

$$g(n) \leq 2^{2^{\dots^{\overbrace{n}}_h}}$$

Frick and Grohe [131] showed that under standard complexity-theoretic assumptions, the function f in Theorem 3.34 cannot have an elementary dependence on $|\varphi|$,

even when restricted to binary trees or 2-colored paths (both treewidth 1). Later, Lampis [201] ruled out elementary dependence on $|\varphi|$ even on uncolored paths, under a different assumption. Observe that uncolored paths are essentially words over a *unary* alphabet.

From Words to Graphs

All results discussed in this section are generalizations of classic results in automata theory. In particular, all of the algorithms are based on *tree automata*, for an introduction to the topic we refer to the book of Downey and Fellows [105]. The classic theorem due to Büchi [58] and Elgot [110] states that a property of words over an alphabet is recognizable by some finite state automaton if and only if it is definable in MSO logic. (See [275] for an accessible and instructive presentation of these results.) Therefore, Courcelle's Theorem (Theorem 3.34) can be viewed as a generalization of one direction of Büchi's and Elgot's theorems to bounded treewidth graphs (namely that definability in MSO implies recognizability). In 1990, Courcelle [85] also conjectured that the other direction (recognizability implies definability) can be generalized to bounded treewidth graphs, showing that it holds for trees. This conjecture remained open until 2016, when it was finally confirmed by Bojańczyk and Mi. Pilipczuk [43].

In complete analogy, we can consider Theorem 3.36 a generalization of the ‘definability implies recognizability’-direction to graphs of bounded clique-width; Bojańczyk et al. [42] also partially confirmed the clique-width analogue of the other direction in the special case of *linear* clique-width.

3.6 Generalized (Distance) Domination Problems

We now discuss a framework that captures many algorithmic problems related to independence and domination in graphs introduced by Telle and Proskurowski [272]. In the (σ, ρ) -*domination problems*, feasible solutions, called (σ, ρ) sets, are vertex sets with constraints on how many neighbors each vertex of the graph has in the set. For a (σ, ρ) set S , the coordinate σ specifies how many neighbors the vertices *inside* S have in S , and the coordinate ρ specifies how many neighbors the vertices *outside* S have in S . The framework generalizes important and well-studied problems such as MAXIMUM INDEPENDENT SET and MINIMUM DOMINATING SET, as well as PERFECT CODE, MINIMUM SUBGRAPH WITH MINIMUM DEGREE d and a multitude of other problems, see Table 3.1. The *locally checkable vertex partitioning* (LCVP) problems are concerned with partitions of the vertex set of the graph with such neighborhood restrictions, and strictly generalize (σ, ρ) problems. For instance, for fixed q , the q -COLORING problem is an LCVP problem. By the work of Bui-Xuan et al. [62] and Belmonte and Vatshelle [17], all these problems are XP-time solvable parameterized by the mim-width of a given branch decomposition of the input graph.

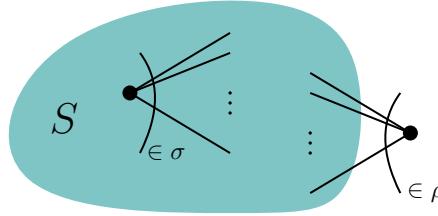


Figure 3.8: Illustration of a (σ, ρ) set S .

We also consider distance versions of problems related to independence and domination, such as DISTANCE- r INDEPENDENT SET. The DISTANCE- r INDEPENDENT SET problem, also studied under the names r -SCATTERED SET and r -DISPERSION (see e.g. [10] and the references therein), asks to find a set of at least k vertices whose pairwise distance is strictly greater than r . Agnarsson et al. [2] pointed out that this problem is equal to the INDEPENDENT SET problem on the r -th power graph G^r of the input graph G .

Generalizing the (σ, ρ) -domination and LCVP problems to their distance versions naturally captures DISTANCE- r INDEPENDENT SET. We use a structural result from Chapter 5 to show that all distance versions of (σ, ρ) as well as LCVP problems are XP-time solvable parameterized by the mim-width of a branch decomposition of the input graph that is provided at the input. Namely the result that taking the r -th power of a graph, for any $r \in \mathbb{N}$, increases the mim-width at most by a factor of 2.

3.6.1 (σ, ρ) -Domination

Let σ and ρ be finite or co-finite subsets of the natural numbers $\sigma, \rho \subseteq \mathbb{N}$. For a graph G , a vertex set $S \subseteq V(G)$ is a (σ, ρ) -dominating set, or simply (σ, ρ) set if

- for each vertex $v \in S$ it holds that $|N(v) \cap S| \in \sigma$, and
- for each vertex $v \in V(G) \setminus S$ it holds that $|N(v) \cap S| \in \rho$.

We illustrate the notion of a (σ, ρ) set in Figure 3.8. For instance, a $(\{0\}, \mathbb{N})$ set is an independent set as there are no edges between the vertices in the set, and we do not care about adjacencies between S and $V(G) \setminus S$. For another example, a $(\mathbb{N}, \mathbb{N}^+)$ -set is a dominating set as we require that for each vertex in $V(G) \setminus S$, it has at least one neighbor in S . (Note that all subsets of \mathbb{N} considered are finite or co-finite.)

There are three types of (σ, ρ) -domination problems: minimization, maximization, and existence. In the rest of this thesis, we are mainly concerned with minimization and maximization problems, defined as follows. Let σ and ρ be finite or co-finite sets.

σ	ρ	d	Standard name
$\{0\}$	\mathbb{N}	1	Independent set
\mathbb{N}	\mathbb{N}^+	1	Dominating set
$\{0\}$	\mathbb{N}^+	1	Maximal Independent set
\mathbb{N}^+	\mathbb{N}^+	1	Total Dominating set
$\{0\}$	$\{0, 1\}$	2	Strong Stable set or 2-Packing
$\{0\}$	$\{1\}$	2	Perfect Code or Efficient Dom. set
$\{0, 1\}$	$\{0, 1\}$	2	Total Nearly Perfect set
$\{0, 1\}$	$\{1\}$	2	Weakly Perfect Dominating set
$\{1\}$	$\{1\}$	2	Total Perfect Dominating set
$\{1\}$	\mathbb{N}	2	Induced Matching
$\{1\}$	\mathbb{N}^+	2	Dominating Induced Matching
\mathbb{N}	$\{1\}$	2	Perfect Dominating set
\mathbb{N}	$\{d, d+1, \dots\}$	d	d -Dominating set
$\{d\}$	\mathbb{N}	$d+1$	Induced d -Regular Subgraph
$\{d, d+1, \dots\}$	\mathbb{N}	d	Subgraph of Min Degree $\geq d$
$\{0, 1, \dots, d\}$	\mathbb{N}	$d+1$	Induced Subg. of Max Degree $\leq d$

Table 3.1: Some vertex subset properties expressible as (σ, ρ) sets. Column d shows $d = \max(d(\sigma), d(\rho))$.

MIN- (MAX-) (σ, ρ) -DOMINATION

Input: A graph G

Question: What is the minimum (maximum) size of any (σ, ρ) set in G ?

In Table 3.1 we give several examples of well-studied problems expressible in this framework.

The d -value of a (σ, ρ) problem is a constant which typically affects the runtime of algorithms for (σ, ρ) problems. For a set $\mu \subseteq \mathbb{N}$, the value $d(\mu)$ should be understood as the highest value in \mathbb{N} we need to enumerate in order to describe μ . Hence, if μ is finite, it is simply the maximum value in μ , and if μ is co-finite, it is the maximum natural number *not* in μ (1 is added for technical reasons).

Definition 3.37 (d-value). Let $d(\mathbb{N}) = 0$. For every non-empty finite or co-finite set $\mu \subseteq \mathbb{N}$, let $d(\mu) = 1 + \min(\max\{x \mid x \in \mu\}, \max\{x \mid x \in \mathbb{N} \setminus \mu\})$.

For a given (σ, ρ) problem $\Pi_{\sigma, \rho}$, its d -value is defined as $d(\Pi_{\sigma, \rho}) := \max\{d(\sigma), d(\rho)\}$, see column d in Table 3.1.

In previous work, Bui-Xuan et al. [62] and Belmonte and Vatshelle [17] showed that all (σ, ρ) problems can be solved in time $n^{\mathcal{O}(w)}$ where w denotes the mim-width of a branch decomposition that is provided as part of the input. As we state the runtime bounds more tightly than what can be read from the statements in the

original works, we provide a sketch of the proof of the following result due to [17, 62], mainly focusing on aspects that affect the runtime rather than the correctness of the algorithms.

Theorem 3.38 ([17, 62]). *There is an algorithm that given a graph G and a branch decomposition (T, \mathcal{L}) of G with $w := \text{mimw}(T, \mathcal{L})$ solves each (σ, ρ) problem Π with $d := d(\Pi)$*

- (i) *in time $\mathcal{O}(n^{4+2d \cdot w})$, if T is a caterpillar, and*
- (ii) *in time $\mathcal{O}(n^{4+3d \cdot w})$, otherwise.*

Proof (sketch). For a positive integer d , and a set $A \subseteq V(G)$, we call two subsets $X \subseteq A$ and $Y \subseteq A$ d -neighbor equivalent w.r.t. A , and we write $X \equiv_A^d Y$, if

$$\forall v \in V(G) \setminus A: \min(d, |X \cap N(v)|) = \min(d, |Y \cap N(v)|).$$

We denote by $\text{nec}(\equiv_A^d)$ the number of equivalence classes of \equiv_A^d , and for $X \subseteq A$, by $\text{rep}_A^d(X)$ an equivalence class representative for X .

The crucial insight that makes the dynamic programming algorithm work within the claimed runtime bound is: when tabulating the partial solutions with respect to a node $t \in V(T)$, it suffices to store one optimum partial solution for each pair $(R_t, R_{\bar{t}})$, where R_t is an equivalence class representative for $\equiv_{V_t}^d$ and $R_{\bar{t}}$ is an equivalence class representative for $\equiv_{V_t}^d$. In this way, an upper bound on $\text{nec}(\equiv_A^d)$ gives an upper bound on the number of table entries to be considered.

Now, let ℓ be a leaf of T and $v \in V(G)$ be such that $\mathcal{L}(v) = \ell$. Then, there are two equivalence classes for $\equiv_{\{v\}}^d$, and $d + 1$ equivalence classes for $\equiv_{V(G) \setminus \{v\}}^d$, so we only have $\mathcal{O}(d)$ partial solutions to consider. For an internal node $t \in V(T)$ with children a and b , we can compute the necessary partial solutions in the following way. For each triple $(R_a, R_b, R_{\bar{t}})$ of an equivalence class representative R_a of $\equiv_{V_a}^d$, an equivalence class representative R_b of $\equiv_{V_b}^d$, and an equivalence class representative $R_{\bar{t}}$ of $\equiv_{V_t}^d$, compute $R_t = \text{rep}_{V_t}^d(R_a \cup R_b)$, $R_{\bar{a}} = \text{rep}_{V_a}^d(R_b \cup R_{\bar{t}})$, and $R_{\bar{b}} = \text{rep}_{V_b}^d(R_a \cup R_{\bar{t}})$, and update the table entry for $(R_t, R_{\bar{t}})$ according to the partial solution obtained from combining the partial solution stored for $(R_a, R_{\bar{a}})$ with the one stored for $(R_b, R_{\bar{b}})$.

Let $\text{nec}_d(T, \mathcal{L}) = \max_{t \in V(T)} \max\{\text{nec}(\equiv_{V_t}^d), \text{nec}(\equiv_{V_t}^d)\}$. We argue that for each internal node $t \in V(T)$, all table entries can be computed in time $\mathcal{O}(\text{nec}_d(T, \mathcal{L})^3 \cdot n^3)$. It can be shown [62, Lemma 1] that for a set $A \subseteq V(G)$, a set of canonical equivalence class representatives of \equiv_A^d can be enumerated in time $\mathcal{O}(\text{nec}(\equiv_A^d) \cdot \log(\text{nec}(\equiv_A^d)) \cdot n^2)$, and given a set $X \subseteq A$, one can find a pointer to $\text{rep}_A^d(X)$ in time $\mathcal{O}(\log(\text{nec}(\equiv_A^d)) \cdot |X| \cdot n)$.

The former computation is invoked once, taking time at most $\mathcal{O}(\text{nec}_d(T, \mathcal{L}) \cdot \log(\text{nec}_d(T, \mathcal{L})) \cdot n^2)$. Next, there are at most $\text{nec}_d(T, \mathcal{L})^3$ triples of equivalence class representatives to consider, and the remaining computations take time at most $\mathcal{O}(n^3)$,

applying the latter of the two aforementioned tasks (finding a pointer to an equivalence class representative), and noting that each representative is of size at most $\mathcal{O}(n)$, and that $\log(\text{nec}_d(T, \mathcal{L})) \leq \mathcal{O}(n)$. Therefore, we compute all table entries for $t \in V(T)$ in time at most $\mathcal{O}(\text{nec}_d(T, \mathcal{L})^3 \cdot n^3)$. As $|V(T)| = \mathcal{O}(n)$, and the computation for a leaf node takes constant time, the whole algorithm takes time at most $\mathcal{O}(\text{nec}_d(T, \mathcal{L})^3 \cdot n^4)$.

In [17, Lemma 2] it is shown that $\text{nec}(\equiv_A^d) \leq n^{d \cdot \text{mim}(A)}$. Hence the dynamic programming algorithms that we sketched above run in time $\mathcal{O}(n^{4+3d \cdot w})$, where $w = \text{mimw}(T, \mathcal{L})$, proving item (ii). For item (i), we observe that for an internal node $t \in V(T)$ with children a and b such that b is a leaf node, the number of triples to consider reduces to $\mathcal{O}(\text{nec}_d(T, \mathcal{L})^2)$, as there are only $\mathcal{O}(1)$ many equivalence class representatives to consider for $\equiv_{V_b}^d$. If T is a caterpillar, then each internal node of T is of that shape, implying that in this case, the algorithms run in time $\mathcal{O}(n^{4+2d \cdot w})$. \square

3.6.2 Distance- r (σ, ρ) -Domination

We generalize (σ, ρ) -domination problems to their distance- r version: For a graph G and a vertex $v \in V(G)$, let $N_G^r(v)$, or simply $N^r(v)$, denote the *ball of radius r around v* , i.e.

$$N_G^r(v) := \{w \in V(G) \setminus \{v\} \mid \text{dist}_G(v, w) \leq r\}.$$

Let σ and ρ be finite or co-finite subsets of \mathbb{N} . Then, a vertex set $S \subseteq V(G)$ is called a *distance- r (σ, ρ) -dominating set*, if

- for each vertex $v \in S$ it holds that $|N^r(v) \cap S| \in \sigma$, and
- for each vertex $v \in V(G) \setminus S$ it holds that $|N^r(v) \cap S| \in \rho$.

We associate with these sets minimization, maximization, and existence problems in the natural way.

Recall that for a graph G and an integer r , the *r -th power of G* , denoted by G^r , is the graph that has the same vertex set as G , with each pair of distinct vertices being adjacent if their distance in G is at most r . Later, in Chapter 5, we prove that taking an (arbitrarily high) power of a graph only increases its mim-width by a factor of at most 2. Specifically, we prove the stronger statement that given a graph G with one of its branch decompositions (T, \mathcal{L}) , we have that¹⁰

$$\text{mimw}_{G^r}(T, \mathcal{L}) \leq 2 \cdot \text{mimw}_G(T, \mathcal{L}), \quad (3.4)$$

i.e. we can use the same decomposition for G^r to witness the upper bound. A proof of this is given in Theorem 5.4 on page 90.

Using this inequality and the following simple observation, we can lift the algorithmic results of Theorem 3.38 to the distance versions of all (σ, ρ) problems.

¹⁰Since we use the same branch decomposition both for G and for G^r , we use the subscript to clarify which graph is used to evaluate the width function.

Observation 3.39. For a positive integer r , a graph G and a vertex $u \in V(G)$, the r -neighborhood of u is equal to the neighborhood of u in G^r , i.e. $N_G^r(u) = N_{G^r}(u)$.

In particular, the previous observation shows that solving a distance- r (σ, ρ) problem on G is the same as solving the standard (i.e. distance-1) variation of the problem on G^r . Hence, we may reduce our problem to the standard version by simply computing the graph power. We have the following consequence.

Corollary 3.40. *There is an algorithm that for all $r \in \mathbb{N}^+$, given a graph G and a branch decomposition (T, \mathcal{L}) of G with $w := \text{mimw}(T, \mathcal{L})$ solves each distance- r (σ, ρ) problem Π with $d := d(\Pi)$*

- (i) in time $\mathcal{O}(n^{4+4d \cdot w})$, if T is a caterpillar, and
- (ii) in time $\mathcal{O}(n^{4+6d \cdot w})$, otherwise.

Proof. Let G be the input graph and (T, \mathcal{L}) the provided branch decomposition. We apply the following algorithm:

Step 1. Compute the graph G^r .

Step 2. Solve the standard (distance-1) version of the problem on G^r , providing (T, \mathcal{L}) as the branch decomposition.

Step 3. Return the answer of the algorithm ran in Step 2 without modification.

Computing G^r in Step 1 takes at most $\mathcal{O}(n^3)$ time using standard methods, Step 3 takes constant time. The worst time complexity is hence found in Step 2. By Equation (3.4), the mim-width of (T, \mathcal{L}) on G^r is at most twice the mim-width of the same decomposition on G . The stated runtime then follows from Theorem 3.38. The correctness of this procedure follows immediately from Observation 3.39. \square

3.6.3 Locally Checkable Vertex Partitioning Problems

A generalization of (σ, ρ) problems are the *locally checkable vertex partitioning* (LCVP) problems which we now discuss. A *degree constraint matrix* D is a $q \times q$ matrix where each entry is a finite or co-finite subset of \mathbb{N} . For a graph G and a partition of its vertices $\mathcal{V} = \{V_1, V_2, \dots, V_q\}$ (where some parts of the partition may possibly be empty), we say that it is a D -partition if and only if, for each $i, j \in [q]$ and each vertex $v \in V_i$, it holds that $|N(v) \cap V_j| \in D[i, j]$.

For instance, let D_3 be the 3×3 matrix whose diagonal entries are $\{0\}$ and the non-diagonal ones are \mathbb{N} , i.e.

$$D_3 := \begin{bmatrix} \{0\} & \mathbb{N} & \mathbb{N} \\ \mathbb{N} & \{0\} & \mathbb{N} \\ \mathbb{N} & \mathbb{N} & \{0\} \end{bmatrix}.$$

Then, a graph G admits a 3-coloring if and only if it admits a D_3 -partition. Typically, algorithmic questions associated with LCVP properties are existential.¹¹ Interesting problems which can be phrased in such terms include the H -COVERING and GRAPH H -HOMOMORPHISM problems where H is fixed, as well as q -COLORING, PERFECT MATCHING CUT and more. We refer to [272] for an overview.

We generalize LCVP properties to their distance- r version, by considering the ball of radius r around each vertex rather than just the immediate neighborhood.

Definition 3.41 (Distance- r neighborhood constraint matrix). A distance- r neighborhood constraint matrix D is a $q \times q$ matrix where each entry is a finite or co-finite subset of \mathbb{N} . For a graph G and a partition of its vertices $\mathcal{V} = \{V_1, V_2, \dots, V_q\}$ (where some parts of this partition may be empty), we say that it is a D -distance- r -partition if and only if, for each $i, j \in [q]$ and each vertex $v \in V_i$, it holds that $|N^r(v) \cap V_j| \in D[i, j]$.

We say that an algorithmic problem is a *distance- r LCVP problem* if the property in question can be described by a distance- r neighborhood constraint matrix. For example, the distance- r version of a problem such as q -COLORING can be interpreted as an assignment of at most q colours to vertices of a graph such that no two vertices are assigned the same colour if they are at distance r or closer.

For a given distance- r LCVP problem Π , its d -value $d(\Pi)$ is the maximum d -value over all the sets in the corresponding neighborhood constraint matrix. As in the case of (σ, ρ) problems, combining Equation (3.4) with Observation 3.39 and the works [17, 62] we have the following.

Corollary 3.42. *There is an algorithm that for all $r \in \mathbb{N}^+$, given a graph G and a branch decomposition (T, \mathcal{L}) of G with $w := \text{mimw}(T, \mathcal{L})$ solves each distance- r LCVP problem Π with $d := d(\Pi)$*

- (i) in time $\begin{cases} \mathcal{O}(n^{4+2qd \cdot w}), & \text{for } r = 1 \\ \mathcal{O}(n^{4+4qd \cdot w}), & \text{for } r > 1 \end{cases}$ if T is a caterpillar, and
- (ii) in time $\begin{cases} \mathcal{O}(n^{4+3qd \cdot w}), & \text{for } r = 1 \\ \mathcal{O}(n^{4+6qd \cdot w}), & \text{for } r > 1 \end{cases}$ otherwise.

As we state the runtime bounds in the previous corollary in a different way than in [62] where they have been proved first, we would like to note that this is justified by an argument parallel to the one we provided in the sketch of the proof of Theorem 3.38 presented in this work.

¹¹Note however that each (σ, ρ) problem can be stated as an LCVP problem via the matrix $D_{(\sigma, \rho)} = \begin{bmatrix} \sigma & \mathbb{N} \\ \rho & \mathbb{N} \end{bmatrix}$, so maximization or minimization of some block of the partition can be natural as well.

4

Graph Classes

A *graph class* is a set of graphs that share a common property. In this chapter, we discuss several graph classes that are relevant to this thesis, in particular considering their connection with the width measures that were the subject of the previous chapter. We present the graph classes in several thematic groups, and to each such group we devote one section. Whenever relevant, we illustrate members of these graph classes, and briefly touch upon the complexity of their recognition problems. This chapter culminates in Section 4.7 where we survey some upper and lower bounds on the width of these classes. For more thorough introductions to the field of graph classes, we refer to the classic books of Brandstädt et al. [52] and of Golumbic [147]. To give an example of how the structure imposed by a graph class is exploited in the design of algorithms, we give an algorithm for the MAXIMUM INDEPENDENT SET problem on chordal graphs in Section 4.2.

The definition of a graph class will be given in the following format. In a separate paragraph, we state the name of the class followed by a specification of the property that a graph has to satisfy to belong to this graph class. For instance, the class of *bipartite* graphs that we have seen earlier already, would be introduced as follows.

BIPARTITE. A graph G is called *bipartite* if there is a 2-partition (A, B) of $V(G)$ such that both A and B are independent sets in G .

4.1 Subclasses of Planar Graphs

In this section, we list some basic classes of planar graphs. We call a drawing of a graph G *planar*, if no pair of edges of G crosses in the drawing, see Figure 4.1 for an illustration.

PLANAR. A graph is called *planar* if it has a planar drawing.

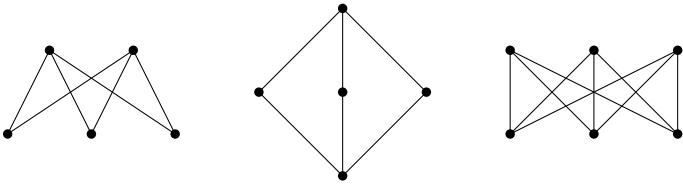


Figure 4.1: A non-planar drawing of a planar graph (left), a planar drawing of the same graph (middle), and a non-planar graph (right). Note that the graph on the left and in the middle is not outerplanar, but the drawing in the middle shows that it is 2-outerplanar. Both graphs are bipartite.

Observe that each planar drawing of a graph partitions the plane into a set of *faces*, where a face can be understood as a region of the plane that can be traversed completely without ‘hopping over’ some edge of the graph. There is one special infinite face, namely the outside that surrounds the drawing of the graph. We refer to this face as the *outer face*.

We introduce several subclasses of planar graphs whose members admit special planar drawings with respect to the outer face.

CACTUS. A graph is called a *cactus* if it has a planar drawing such that all edges are on the same face (the outer face).

Another (possibly more common) way to define the class of cacti is via the property that each edge of a cactus is contained in at most one cycle.

If we require all *vertices* (instead of all edges) to be on the outer face, then we arrive at the class of outerplanar graphs. Observe that each cactus is outerplanar, but not vice versa. For instance, the diamond graph, which is obtained from a C_4 by adding an edge between two non-adjacent vertices, is outerplanar but not a cactus.

OUTERPLANAR. A graph is called *outerplanar* if it has a planar drawing such that all vertices are on the same face (the outer face).

Another way of viewing an outerplanar drawing of a graph is the following: if we remove all vertices that lie on the outer face, then no vertex remains. If we cannot obtain an empty graph in one such step, but in k steps, then we talk about a k -*outerplanar* graph.

k -OUTERPLANAR. If a graph is outerplanar, then we call it 1-outerplanar. For $k \geq 2$, a graph is called k -outerplanar, if it admits a planar drawing such that removing all vertices on the outer face results in a $(k - 1)$ -outerplanar graph.

We observed the following inclusions between these graph classes. (As a convention, we regard the empty graph to be a k -outerplanar graph for any $k \in \mathbb{N}$.)

$$\begin{aligned} \text{FOREST} &\subsetneq \text{CACTUS} \subsetneq \text{OUTERPLANAR} \subsetneq \text{2-OUTERPLANAR} \\ &\subsetneq \text{3-OUTERPLANAR} \subsetneq \cdots \subseteq \text{PLANAR} \end{aligned}$$

4.2 Chordal Graphs

The class of chordal graphs is among the most commonly studied and well understood classes. Recall that a *chord* in a cycle C contained in some graph G is an edge in G between two vertices on C that are not adjacent in C ; i.e. it is an edge $uv \in E(G)$ where $u, v \in V(C)$ and $uv \notin E(C)$.

CHORDAL. A graph G is called a *chordal graph* if every cycle in G of length at least four has a chord.

Chordal graphs have several characterizations that are very helpful in the design of algorithms. A vertex v in a graph G is called *simplicial* if $N_G(v)$ is a clique. Dirac [102] showed that each chordal graph has a simplicial vertex. Together with the observation that the class of chordal graphs is closed under taking induced subgraphs (i.e. removing vertices from a chordal graph results in a chordal graph), this implies that each chordal graph G admits what is called a *perfect elimination ordering (peo)*: an ordering v_1, \dots, v_n of the vertices of G such that for all $i \in [n-1]$, v_i is a simplicial vertex in $G[\{v_{i+1}, \dots, v_n\}]$. Fulkerson and Gross [132] showed that a graph is chordal *if and only if* it has a perfect elimination ordering.

Another way of characterizing chordal graphs is as ‘fat trees’. Concretely, chordal graphs are precisely the graphs that have a tree decomposition (recall Definition 3.1 on page 16) in which each bag is a maximal clique [63, 137, 285]. This special type of tree decomposition is called a *clique tree*. Since we use it in the algorithmic example of the MAXIMUM INDEPENDENT SET problem on chordal graphs below, we define it formally.

Definition 4.1 (Clique Tree). Let G be a graph. A *clique tree* of G is a tree decomposition (T, \mathcal{B}) such that for all $t \in V(T)$, B_t is a maximal clique in G .

Before we proceed, let us observe that the number of nodes in a clique tree is at most the number of vertices of the underlying graph. Since each bag of a clique tree is a maximal clique, it suffices to argue that each chordal graph G on n vertices has at most n maximal cliques. Consider the perfect elimination ordering v_1, \dots, v_n of G . For each maximal clique in G , consider its leftmost vertex v_i in the peo. Since v_i is simplicial in $G[\{v_{i+1}, \dots, v_n\}]$, its neighborhood in $G[\{v_{i+1}, \dots, v_n\}]$ is a clique, and therefore v_i cannot be the leftmost vertex of any other maximal clique in G . This means that each vertex in G is the leftmost vertex of at most one maximal clique in G , and therefore the number of maximal cliques in G is no more than n .

Observation 4.2. Let G be an n -vertex chordal graph and let (T, \mathcal{B}) be a clique tree of G . Then, $|V(T)| \leq n$.

We now consider the MAXIMUM INDEPENDENT SET problem on chordal graphs.

MAXIMUM INDEPENDENT SET/CHORDAL

Input: A chordal graph G .

Question: What is the size of a maximum independent set in G ?

Blair and Peyton [26] gave a linear-time algorithm that given a chordal graph G , constructs a clique tree (T, \mathcal{B}) of G . We might be tempted to immediately apply the treewidth-based algorithm from Section 3.1 using the tree decomposition (T, \mathcal{B}) . However, its runtime depends *exponentially* on the maximum size of any bag in \mathcal{B} , and this quantity is not bounded in the case of chordal graphs. Note that a complete graph on n vertices is a chordal graph, and its clique tree consists of a single bag containing all its vertices.

Instead, we can exploit the property that each bag of \mathcal{B} is a clique: this immediately implies the following observation.

Observation 4.3. Let G be a chordal graph with clique tree (T, \mathcal{B}) . Then, for each $t \in V(T)$ and each independent set I of G , $|I \cap B_t| \leq 1$.

Assume that T is rooted, and recall that for each $t \in V(T)$, T_t is the subtree of T rooted at t , that $V_t = \bigcup_{s \in V(T_t)} B_s$, and $G_t = G[V_t]$. The previous observation suggests a dynamic programming algorithm where for each $t \in V(T)$ and each subset X of B_t of size at most one, we store the maximum size of any independent set in G_t whose intersection with B_t equals X .

Definition of the table entries. For each $t \in V(T)$, and each $X \subseteq B_t$ with $|X| \leq 1$, we let $\text{tab}[t, X] = \max\{|I| : I \text{ is an independent set in } G_t \text{ and } I \cap B_t = X\}$. \triangleleft

As before, we compute the table entries in a bottom-up manner, starting in the leaves of T , and then moving on to the internal nodes.

Leaves of T . If $t \in V(T)$ is a leaf of T , then for each $X \subseteq B_t$ with $|X| \leq 1$, we simply let $\text{tab}[t, X] := |X|$. \triangleleft

Internal nodes of T . Suppose $t \in V(T)$ is an internal node of T with children s_1, \dots, s_d , and that the table entries at the children have been computed. For each $X \subseteq B_t$ with $|X| \leq 1$, we want to determine the maximum size of any independent set in G_t whose intersection with B_t is equal to X . Such an independent set is formed by taking the union over all children s_i of a largest independent set in G_{s_i} that is in some sense *compatible* with the choice of X . It suffices to only consider the intersection X_i^* of independent sets in G_{s_i} with the vertices in B_{s_i} .

Concretely, compatible means the following. First, suppose that $X = \emptyset$. Then, X_i^* cannot contain any vertex from $B_t \cap B_{s_i}$. Therefore we look for the subset $X_i^* \subseteq B_{s_i} \setminus B_t$ of size at most one such that $\text{tab}[s_i, X_i^*]$ is maximum among all such choices. If $X \neq \emptyset$, then $X = \{v\}$ for some $v \in B_t$. Now, if $v \in B_{s_i}$, then clearly X_i^* must be equal to $\{v\}$. If $v \notin B_{s_i}$, then we can choose another subset of B_{s_i} . However,

this subset cannot contain any neighbors of v ,¹ as otherwise the resulting set would have an edge between v and the vertex from B_{s_i} . In other words, we cannot pick any vertex from $N(v) \cap B_{s_i}$ which by the properties of a clique tree is equal to $B_t \cap B_{s_i}$. Therefore, as in the first case, we look for the subset $X_i^* \subseteq B_{s_i} \setminus B_t$ of size at most one such that $\text{tab}[s_i, X_i^*]$ is maximized. To summarize, for each $X \subseteq B_t$ with $|X| \leq 1$, we find for each $i \in [d]$,

$$X_i^* := \begin{cases} \operatorname{argmax}_{X_i} \{\text{tab}[s_i, X_i] \mid X_i \subseteq B_{s_i} \setminus B_t, |X_i| \leq 1\}, & \text{if } X = \emptyset \\ & \text{or } X \neq \emptyset \text{ and } X \not\subseteq B_{s_i} \\ X, & \text{if } X \neq \emptyset \text{ and } X \subseteq B_{s_i} \end{cases}$$

and update $\text{tab}[t, X] := \sum_{i \in [d]} \text{tab}[s_i, X_i^*]$. \triangleleft

We skip a formal proof that the above algorithm computes the table entries correctly. Once the table entries at the root \mathbf{r} of T have been computed, the solution to the instance can be found as $\max_{X \subseteq B_{\mathbf{r}}, |X| \leq 1} \text{tab}[\mathbf{r}, X]$, since $G_{\mathbf{r}} = G$. Regarding the runtime of the algorithm, recall that $|V(T)| = \mathcal{O}(n)$ by Observation 4.2, and that at each node of T , there are at most $\mathcal{O}(n)$ table entries to compute. In the leaves, computing a table entry takes constant time, and at an internal node, we spend at most $\mathcal{O}(n)$ time. Since a clique tree of the input graph can be constructed in $\mathcal{O}(n + m) = \mathcal{O}(n^2)$ time [26], we have the following theorem.

Theorem 4.4. *There is an algorithm that given an n -vertex chordal graph G , computes in time $\mathcal{O}(n^3)$ the size of a maximum independent set in G .*

We remark that there are faster, i.e. linear-time algorithms for MAXIMUM INDEPENDENT SET on chordal graphs that make direct use of the perfect elimination ordering, see e.g. [137, 258].

We conclude this section by defining several subclasses of chordal graphs that will be important later. Given a cycle C in a graph G , an *odd chord* in C is a chord between vertices u, v in C that are at odd distance in C .

STRONGLY CHORDAL. A graph G is called *strongly chordal* if it is chordal and every cycle of even length at least six has an odd chord.

We illustrate the graph classes from the previous and the following definition in Figure 4.2.

SPLIT. A graph G is called a *split graph* if there is a 2-partition (C, I) of $V(G)$ such that C is a clique in G and I is an independent set in G .

To see that each split graph is indeed chordal, let G be a split graph with 2-partition (C, I) such that C is a clique and I an independent set. Without loss of generality, assume that C is a maximal clique. We can build a clique tree of G as

¹Note that by the separator property of tree decompositions, since $v \notin B_{s_i}$, we know that each neighbor of v in G_{s_i} is also contained in B_{s_i} .

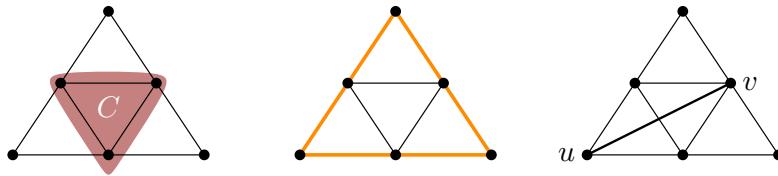


Figure 4.2: On the left, a split graph with its clique C marked. As shown in the middle, this graph is not strongly chordal as it contains a cycle on six vertices that has no odd chord. Adding an edge between u and v as shown on the right side makes the graph strongly chordal, since uv is an odd chord in that cycle (which is the only cycle of length at least six in the given graph).

follows. First, we add a bag B_C containing C . Then, for each $v \in I$, we add a bag containing $N_G[v]$, and make it adjacent to B_C . It is straightforward to verify that this is a clique tree of G . Note that this clique tree has the shape of a star.

The next class we consider also admits clique trees of a very special form, namely in which each pair of adjacent (in the tree) maximal cliques shares precisely one vertex. This means in turn that such graphs consist of a collection of (maximal) cliques that are glued together along cut-vertices in a tree-like way.

BLOCK. A graph is called a *block graph* if each of its maximal 2-connected components (its *blocks*) is a clique.

Lastly, we consider a subclass of chordal graphs that is of importance in the field of phylogenetic studies, i.e. the reconstruction of ancestral relations in biology, see e.g. [64].

LEAF POWER. Let $r \in \mathbb{N}$. A graph G is an *r -leaf power* if there exists a tree T , called a leaf root, whose leaf set is $V(G)$ such that $uv \in E$ if and only if the distance between u and v in T is at most r . A graph is called a *leaf power* if it is an r -leaf power for some $r \in \mathbb{N}$.

4.3 Intersection Graphs

Intersection graphs are a very popular ‘meta graph class’. The theme is that we are given a set of objects over a universe, and we create a corresponding graph by adding one vertex per object and an edge between two vertices if their corresponding objects overlap. The formal definition is as follows.

INTERSECTION GRAPH. A graph G is called an *intersection graph* if there is some universe U and a family of subsets \mathcal{S} over U such that

$$V(G) = \mathcal{S} \text{ and } E(G) = \{\{R, S\} \mid R, S \in \mathcal{S} \wedge R \cap S \neq \emptyset\}.$$

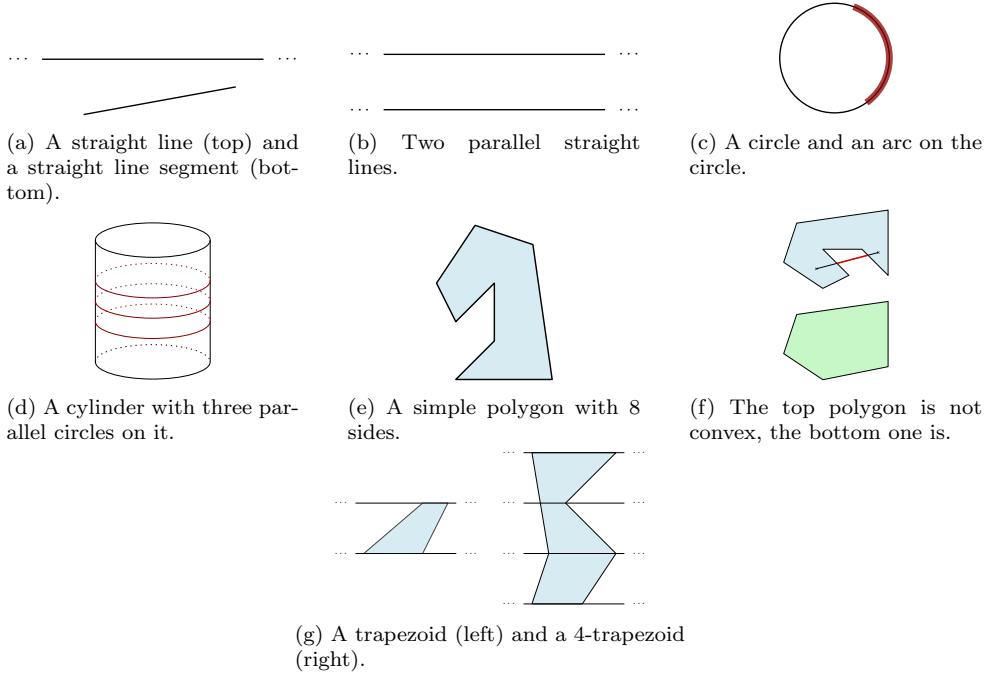


Figure 4.3: Some basic geometric objects that are used in the definitions of the intersection graphs of this section. A polygon is *simple* if it does not intersect itself and if it has no holes. All polygons considered here are simple.

In this case we say that G is the *intersection graph* of \mathcal{S} , and that \mathcal{S} is the *intersection model* of G .

Without imposing some restrictions on the universe U and/or on the set \mathcal{S} , the notion of an intersection graph is not very meaningful; we see below (Observation 4.7) that *any* graph is an intersection graph. In the following, we list some graph classes that are intersection graphs of some low-dimensional geometric objects. Rather than formally defining the basic objects that are used in the following definitions, we illustrate them in Figure 4.3. For an illustration of the graph classes discussed in this section, see Figure 4.4. In Table 4.1 we furthermore give references regarding the complexity of recognizing the members of each graph class.

INTERVAL. A graph G is called an *interval graph* if it is the intersection graph of a set of intervals in \mathbb{R} .

It is not difficult to see that each interval graph G is chordal. We can traverse the intersection model of an interval graph ‘from left to right’ and remember the maximal cliques of G we encountered in this order. This directly translates to a clique tree

(in fact a clique *path*) of G .

Also chordal graphs are famously characterized as intersection graphs. By a classic theorem [63, 137, 285], chordal graphs are precisely the intersection graphs of collections of subtrees of a tree. This connection is directly observed via the clique tree of a chordal graph: If G is a chordal graph with clique tree (T, \mathcal{B}) , then for each $v \in V(G)$, we consider a subtree $T(v)$ of T that consists of all nodes whose bags contain v . Since each bag in \mathcal{B} is a clique, we have that G is the intersection graph of $\{T(v) \mid v \in V(G)\}$.

The members of the next graph class we consider can be obtained in such a way, with two additional restrictions that need to be fulfilled: the tree has to be *rooted* and the models of the vertices are *paths*. For the following definition, a *rooted directed tree* is a rooted tree in which all edges are directed from the parent vertex to the child vertex.

ROOTED DIRECTED PATH. A graph is called a *rooted directed path graph* if it is the intersection graph of directed paths in a rooted directed tree.

Rooted directed path graphs are not only chordal but *strongly* chordal. Brandstädt et al. [51] even showed that each rooted directed path graph is a leaf power graph, which will be important for us later. On the other hand, rooted directed path graphs generalize interval graphs: We can represent the real line by a sufficiently long directed path, and the interval models of the vertices by subpaths of that directed path, therefore

$$\text{INTERVAL} \subseteq \text{ROOTED DIRECTED PATH} \subseteq \text{LEAF POWER}.$$

The remaining graph classes we discuss in this section are not chordal. It is not difficult to construct a model of C_4 , a cycle on four vertices, according to the definition of each of the following classes.

First, we consider a graph class that is essentially obtained by using ‘intervals’ of a circle instead of the (real) line. Such ‘intervals’ are known as (*circular*) *arcs*, and the resulting graph class strictly generalizes interval graphs.

CIRCULAR ARC. A graph G is called a *circular arc graph* if it is the intersection graph of a set of arcs of a circle.

PERMUTATION. A graph G is called a *permutation graph* if it is the intersection graph of a set of straight line segments between two parallel straight lines.

To get a better understanding of where the term permutation graph comes from, we explain an alternative representation of a permutation graph and how it is connected to the definition given above.

For each permutation graph G on vertices v_1, \dots, v_n , there is a permutation $\pi: [n] \rightarrow [n]$ such that for all $i < j$, v_i is adjacent to v_j in G if and only if $\pi(i) > \pi(j)$, i.e. the permutation π swaps the relative order of i and j . Now, we can use such

Graph Class	Complexity of Recognition
INTERVAL	$\mathcal{O}(n+m)$ [48, 152]
CIRCULAR ARC	$\mathcal{O}(n+m)$ [218, 178]
PERMUTATION	$\mathcal{O}(n+m)$ [219]
CIRCULAR PERMUTATION	$\mathcal{O}(n+m)$ [271]
CIRCLE	$\mathcal{O}(n^2)$ [270]
k -POLYGON	NP-c for unbounded k , $\mathcal{O}(4^k \cdot n^2)$ [112]
(2-)TRAPEZOID	$\mathcal{O}(n^2)$ [211], see also [221]
k -TRAPEZOID	NP-c for fixed $k \geq 3$ [38, 288], see also [17]
CIRCULAR k -TRAPEZOID	open for any $k \geq 2$
H -GRAPHS	NP-c for fixed H if H is not a cactus [72]
	$\mathcal{O}(n^{ E(H) ^2})$ if H is a tree [72]
ROOTED DIRECTED PATH	$\mathcal{O}(n+m)$ [138]
TOLERANCE	NP-c [222]
BOUNDED TOLERANCE	NP-c [222]

Table 4.1: Complexity of the recognition problem of several intersection graph classes. By n we denote the number of vertices of the input graph, and by m the number of its edges. All recognition algorithms we mention here are *constructive*, meaning that they produce an intersection model if the input graph belongs to the class.

a permutation π to obtain an intersection model of G in the sense of the previous definition as follows. Take two parallel lines, in the following referred to as the *lower* and the *upper* line. On the lower line, label n consecutive points $1, 2, \dots, n$. On the upper line, label n consecutive points with $\pi(1), \pi(2), \dots, \pi(n)$. For each $i \in [n]$, connect the point labeled i on the lower line with the point labeled i on the upper line by a straight line segment (call it the i -th line segment). Then, the i -line segment is a model for the vertex v_i : for all $j \neq i$, we have that v_i and v_j are adjacent if and only if the relative order of i and j is swapped by π if and only if the i -th line segment and the j -th line segment intersect.

Next, we consider a circular variant of permutation graphs. The intersection models of these graphs can be imagined as follows. Take two parallel straight lines and glue them on the surface of a cylinder, identifying the two endpoints of each line such that each of them forms a circle. Then draw straight line segments between the resulting two parallel cycles, and take their intersection graph.

CIRCULAR PERMUTATION. A graph G is called a *circular permutation graph* if it is the intersection graph of a set of straight line segments between two parallel circles on a cylinder.

We now consider some graph classes that are intersection graphs of straight line segments in ‘enclosing shapes’ that are different from two parallel straight lines. First, we consider k -polygons, and then circles.

k -POLYGON. For $k \geq 3$, a graph is a k -*polygon graph* if it is the intersection graph of straight line segments between two distinct sides in a convex polygon with k sides.

CIRCLE. A graph is a *circle graph* if it is the intersection graph of straight line segments between points on a circle.

As k increases, the shape of a (regular) polygon with k sides converges to that of a circle. Elmallah and Stewart showed that indeed, the union over $k = 3, 4, \dots$ of all k -polygon graphs is equal to the class of circle graphs.

Theorem 4.5 (Elmallah and Stewart [111]). CIRCLE = $\bigcup_{k=3}^{\infty}$ k -POLYGON.

As we see later, it is necessary to let k converge to infinity in the previous theorem, meaning that there is no finite $K \in \mathbb{N}$ such that CIRCLE = $\bigcup_{k=3}^K$ k -POLYGON: for finite k , each k -polygon graph has bounded mim-width while the class of circle graphs has unbounded mim-width.

In the intersection model of the next class, we consider simple 2-dimensional objects instead of (1-dimensional) lines. In particular, we consider trapezoids, and their generalizations k -trapezoids.

Definition 4.6 ((k -)Trapezoid). Let $k \geq 2$, and let $\mathcal{L} := (L_1, \dots, L_k)$ be parallel straight lines. A k -*trapezoid* T (in \mathcal{L}) is a polygon with (at most) $2k$ sides that are formed by taking two points t_i^1, t_i^2 on each line L_i , and

- connecting t_1^1 and t_1^2 by a straight line segment,
- connecting t_k^1 and t_k^2 by a straight line segment, and
- for $i \in [k-1]$, connecting $\{t_i^1, t_i^2\}$ and $\{t_{i+1}^1, t_{i+1}^2\}$ by two straight non-crossing line segments.

A 2-trapezoid is simply called a *trapezoid*.

The following graph classes are intersection graphs of k -trapezoids, drawn on different surfaces.

k -TRAPEZOID. Let $k \geq 2$. A graph is called a k -*trapezoid graph* if it is the intersection graph of a set of k -trapezoids in a set of k parallel straight lines. A 2-trapezoid graph is simply called *trapezoid graph*.

CIRCULAR k -TRAPEZOID. Let $k \geq 2$. A graph is called a *circular k -trapezoid graph* if it is the intersection graph of a set of k -trapezoids in a set of k parallel cycles on a cylinder. A circular 2-trapezoid graph is simply called a *circular trapezoid graph*.

The last class we consider here is not an intersection graph class of geometric objects, but of connected subgraphs of some other graph.

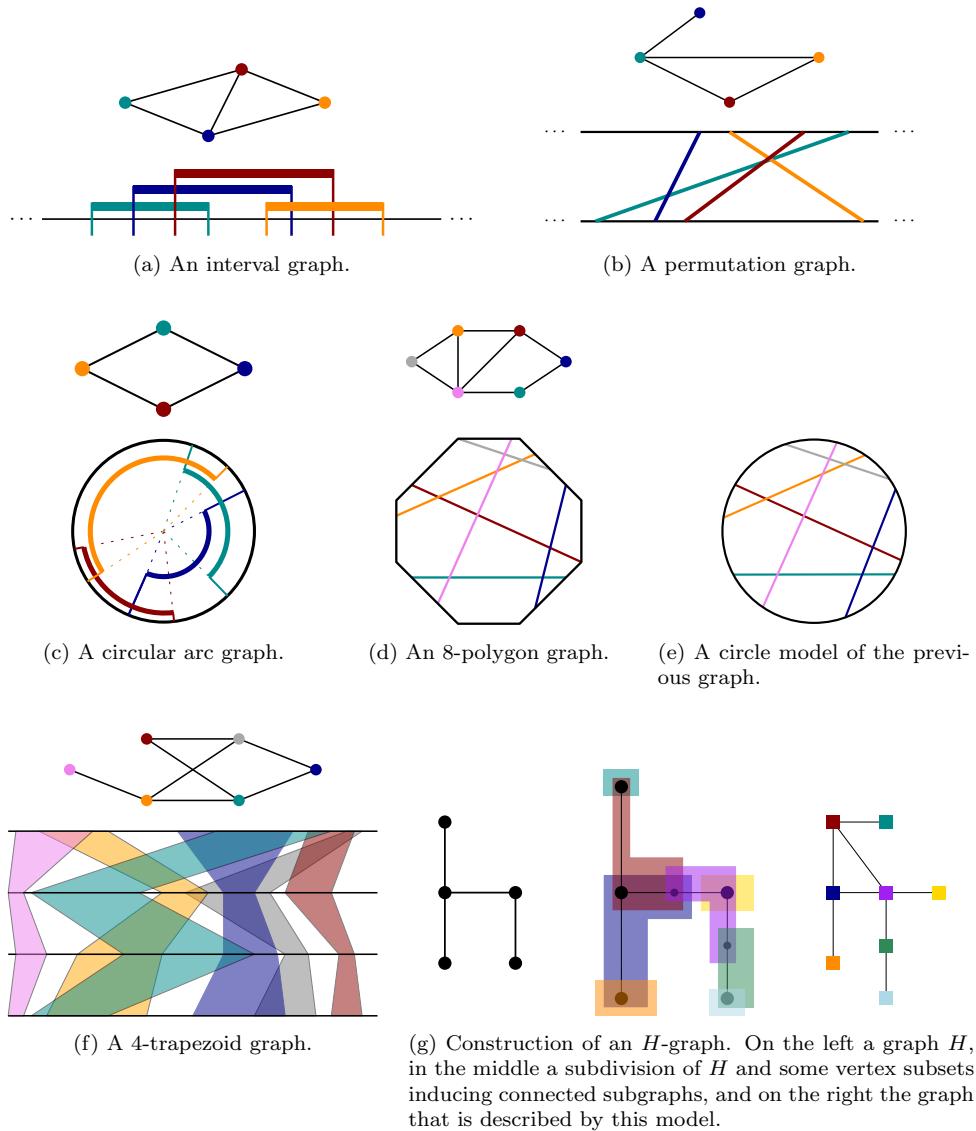


Figure 4.4: Examples of the intersection graphs defined in this section. For *circular* permutation/ k -trapezoid graphs, imagine a representation such as the one shown for permutation/ k -trapezoid graphs, but wrapped around a cylinder, and gluing the ends of the lines together.

H-GRAPH. Let H be a graph. A graph G is called an *H-graph* if there is a subdivision H' of H such that G is the intersection graph of a set of vertex subsets of H' that induce connected subgraphs.

We make an immediate observation from the definition of *H*-graphs that will be useful in later proofs. For a graph H , we can construct a *H*-representation of itself: We subdivide each edge of H once to obtain H' . Then, we create an *H*-representation \mathcal{M} of H by adding for each vertex $v \in V(H)$ a model $M_v := N_{H'}[v]$.

Observation 4.7. Any graph H is an *H*-graph.

H-graphs naturally capture two of the graph classes we have seen in this section: interval graphs are precisely the K_2 -graphs, and circular arc graphs are the K_3 -graphs. Perhaps in slight dissonance with Observation 4.7 (which is there for technical reasons), we do not consider chordal graphs to be *H*-graphs, since we usually consider *H*-graphs for *fixed* graphs H . Moreover, the size of H is important in the applications of *H*-graphs in the remainder of this work.

Tolerance Graphs

The graph classes we discuss here are not intersection graphs but so closely related to interval graphs that we decided to present them here, at a slight distance from the other classes defined in this section. Instead of the *intersection* of intervals, these graph classes are defined with respect to the *size of the overlap* of intervals.

TOLERANCE. A graph G is called a *tolerance graph*, if we can associate with each vertex $v \in V(G)$ a closed interval I_v in \mathbb{R} and a tolerance $t_v \in \mathbb{R}$, such that for each pair of vertices $u, v \in V(G)$,

$$uv \in E(G) \Leftrightarrow |I_u \cap I_v| \geq \min\{t_u, t_v\}.$$

We call $\{(I_v, t_v)\}_{v \in V(G)}$ a *tolerance representation* of G .

In the definition of the following subclass of tolerance graphs, we additionally impose a restriction on the value of the tolerance of each vertex.

BOUNDED TOLERANCE. A graph G is a *bounded tolerance graph* if it has a tolerance representation $\{(I_v, t_v)\}_{v \in V(G)}$ where for all $v \in V(G)$, $t_v \leq |I_v|$.

4.4 Recursively Generated Graphs (and Relatives)

We now introduce several graph classes that can be constructed by repeatedly inserting new vertices in a structured way. Recall that a vertex is called *isolated* if it has no neighbors, and *universal*, if it is adjacent to all other vertices.

THRESHOLD. A graph G is called a *threshold graph* if it can be obtained from the empty graph by repeatedly adding isolated or universal vertices.

The name *threshold* graph comes from the following equivalent definition (see for instance the book [213]): a graph G is a threshold graph if there is some real number $t \in \mathbb{R}$ (the threshold), and an assignment $a: V(G) \rightarrow \mathbb{R}$ of real numbers to the vertices of G such that for all $u, v \in V(G)$, $uv \in E(G)$ if and only if $a(u) + a(v) \geq t$.

Given a threshold graph G , we can order its vertices v_1, \dots, v_n in such a way that for each $i \in [n]$, the vertex v_i is either isolated or universal in $G[\{v_1, \dots, v_i\}]$. Such an ordering can be viewed as a linear decomposition of G , where for all $i \in [n - 1]$, between position i and $i + 1$ we either take the disjoint union between $G[\{v_1, \dots, v_i\}]$ and $\{v_{i+1}\}$, or a *complete join*, i.e. we add all edges that have one endpoint in the first and the other in the second (single-vertex) graph. The graph class we introduce next is obtained by generalizing this decomposition technique to a *non-linear* variant.

COGRAPH. A graph is called a *cograph* if it is either a single vertex, or the disjoint union or the complete join of two cographs.

Another way of obtaining a (connected component of a) cograph is from a single-vertex graph by repeatedly introducing *true or false twins*. That is, from a connected cograph G we can obtain another connected cograph by considering one of its vertices $u \in V(G)$, and adding a new vertex x such that x is adjacent either to $N_G[u]$ (in this case x is a *true twin* of u), or to $N_G(u)$ (in this case x is a *false twin* of u). If additionally, we allow the operation of introducing a *pendant vertex*, i.e. a new vertex that is adjacent to only one vertex of G , then we obtain the class of *distance-hereditary graphs* [13]. We give the more standard definition based on another property, which is more faithful to its name: that taking connected induced subgraphs preserves the distances between vertices.

DISTANCE-HEREDITARY. A connected graph G is called *distance-hereditary* if for each connected subgraph H of G and all $u, v \in V(H)$, $\text{dist}_G(u, v) = \text{dist}_H(u, v)$.

Another way of generalizing cographs is via their characterization as P_4 -free graphs, i.e. the graphs that do not contain a path on four vertices as an induced subgraph. Various graph classes that are obtained from a relaxation of the absence-requirement of induced P_4 s have received attention in the literature. We list some of them that appear later in this work.

P_4 -SPARSE. A graph G is called P_4 -sparse if each subset of five vertices of G contains at most one induced P_4 .

(q, t) -GRAPH. Let $q \geq 5$ and $t \geq 1$. A graph G is called a (q, t) -graph if each subset of q vertices of G contains at most t distinct induced P_4 s.

Observe that P_4 -SPARSE = $(5, 1)$ -GRAPH, and that (q, t) -GRAPH $\subseteq (q', t')$ -GRAPH whenever $q \geq q'$ and $t \leq t'$. Let G be a graph and $S \subseteq V(G)$ be a set of vertices that

induces a P_4 . A *partner* of S is a vertex $v \in V(G) \setminus S$ such that $S \cup \{v\}$ induces at least two P_4 s.

P_4 -TIDY. A graph G is called P_4 -tidy if any induced P_4 in G has at most one partner.

PARTNER-LIMITED. A graph G is called *partner-limited* if any induced P_4 in G has at most two partners.

4.5 Graphs Related to Orderings

We now consider graphs that are defined (in various ways) in terms of orderings that involve the neighborhoods of vertices. First, for each $d \in \mathbb{N}$, we consider graphs G that admit a linear ordering v_1, \dots, v_n of their vertices such that for each $i \in [n-1]$, v_i has at most d neighbors among the vertices in $\{v_{i+1}, \dots, v_n\}$. These graphs are known as *d-degenerate*, and can equivalently be defined as follows.

d-DEGENERATE. Let $d \in \mathbb{N}$. A graph G is called *d-degenerate*, if each subgraph of G contains a vertex of degree at most d .

In fact, *complements* of *d-degenerate* graphs are more important for the remainder of this thesis.

Co-d-DEGENERATE. Let $d \in \mathbb{N}$. A graph G is called *co-d-degenerate* if its complement \overline{G} is *d-degenerate*.

Next, we consider some subclasses of bipartite graphs. Let G be a bipartite graph with bipartition (A, B) . An ordering a_1, \dots, a_{n_A} of A is said to have the *adjacency property* if for all vertices $b \in B$ that are not isolated, the vertices of $N(b)$ appear consecutively, i.e. for some $i < j \leq n_A$, $N(b) = \{a_i, a_{i+1}, \dots, a_j\}$.

CONVEX/BICONVEX. A bipartite graph is called *convex/biconvex* if at least one/both of the parts of its vertex bipartition admit(s) a linear ordering that has the adjacency property.

BIPARTITE CHAIN. A bipartite graph G with vertex bipartition (A, B) is called a *bipartite chain graph* if there is an ordering a_1, \dots, a_{n_A} of A such that for all $i \in [n-1]$, $N_G(a_i) \subseteq N_G(a_{i+1})$.

We can observe that each ordering as described in the previous definition has the adjacency property. Moreover, if the set A can be linearly ordered according to neighborhood inclusion, then so can B . On the other hand, a graph on four vertices consisting of two isolated edges ($2K_2$) is biconvex but not bipartite chain. Therefore, **BIPARTITE CHAIN \subsetneq BICONVEX**.

We consider a graph class that is also defined in terms of a comparability criterion according to neighborhood inclusion, but in this case we do not restrict ourselves to

Graph Class	Complexity of Recognition
(Co)- d -DEGENERATE	$\mathcal{O}(n + m)$ [217]
CONVEX/BICONVEX	$\mathcal{O}(n + m)$ [52]
BIPARTITE CHAIN	$\mathcal{O}(n + m)$ [157]
DILWORTH- k	$\mathcal{O}(k^2 \cdot n^2)$ [121]
(Co)-COMPARABILITY	$\mathcal{O}(n^\omega)$ [146, 227]

Table 4.2: Complexity of the recognition problem of several graph classes that appeared in Section 4.5. By n we denote the number of vertices of the input graph, by m the number of its edges, and $\omega < 2.3729$ denotes the degree of the polynomial in the runtime of the fastest-known algorithm for matrix multiplication [279, 135].

bipartite graphs, and we measure the largest size of a set of pairwise *incomparable* vertices.

DILWORTH- k . Let G be a graph. Two vertices $u, v \in V(G)$ are called *comparable* if either $N_G(u) \subseteq N_G[v]$ or $N_G(v) \subseteq N_G[u]$. The *Dilworth-number* of G is the size of a largest set of pairwise incomparable vertices. Let $k \in \mathbb{N}$. A graph G is a *Dilworth- k graph* if its Dilworth-number is at most k .

The next graph class is defined in terms of partial orders that a priori do not consider any property of a graph. Let (Ω, \prec) be any strictly² partially ordered set. Then we can associate with (Ω, \prec) a directed acyclic graph D on vertex set Ω , where for each pair $x, y \in \Omega$, the arc (x, y) is present in D if and only if $x \prec y$. The next graph class consists of the undirected graphs that can be obtained in this way by forgetting the directions of the arcs.

COMPARABILITY. A graph G is called a *comparability graph* if there is a strict partial order \prec of $V(G)$ such that for all $u, v \in V(G)$, $uv \in E(G)$ if and only if either $u \prec v$ or $v \prec u$.

Since complements of comparability graphs are important later, we also introduce them here.

Co-COMPARABILITY. A graph G is called a *co-comparability graph* if \overline{G} is a comparability graph.

4.6 Forbidden Induced Subgraph Characterizations

Many graph classes \mathcal{C} have a characterization by a list of *forbidden induced subgraphs* (often called an *obstruction set*). That is, a list $\text{Forb}_{\mathcal{C}}$ of graphs such that a graph G belongs to \mathcal{C} if and only if it does not contain any graph from $\text{Forb}_{\mathcal{C}}$ as an induced

²Meaning that there is no pair $x, y \in \Omega$ such that $x \prec y$ and $y \prec x$.

subgraph. We illustrate obstruction sets for several graph classes we have encountered in this section in Figure 4.5.

This does not only increase structural insight into the graph class at hand, but often it also helps algorithmically. If $\text{Forb}_{\mathcal{C}}$ is a finite list of finite graphs, this immediately implies a polynomial-time recognition algorithm for \mathcal{C} : given a graph G , we can simply try for each graph in $K \in \text{Forb}_{\mathcal{C}}$, and each size- $|V(K)|$ subset S of $V(G)$, whether or not S induces K in G . If the answer is positive for any such test, we can refute membership in \mathcal{C} , and if all tests fail, we can confirm that $G \in \mathcal{C}$.

Even if $\text{Forb}_{\mathcal{C}}$ is infinite, we may still be able to give a polynomial-time recognition algorithm based on $\text{Forb}_{\mathcal{C}}$. For instance, chordal graphs are the $\{C_k \mid k \geq 4\}$ -free graphs (by definition). By brute force, we can test in polynomial time whether a graph contains a C_4 , and there is a polynomial-time algorithm [233] that given a graph G decides if G contains an induced cycle on at least five vertices.

Moreover, in *graph modification problems* [91, 215, 228] we are given a graph G and a graph class \mathcal{C} , and the question is if we can obtain a member of \mathcal{C} by performing a small number of operations (such as vertex deletion, edge deletion/addition, edge contraction) on G . In many instances, forbidden induced subgraph characterizations are key in the design of algorithms for such problems.

In some cases, the name of a class is derived from the forbidden induced subgraph(s) it has. Arguably one of the most famous examples is the following.

CLAW-FREE. A graph is called *claw-free* if it does not have a star with three leaves (a.k.a. $K_{1,3}$ or a *claw*) as an induced subgraph.

Generally, for any (finite or infinite) set of graphs \mathcal{H} , we can define the corresponding graph class that forbids graphs in \mathcal{H} .

\mathcal{H} -FREE. Let \mathcal{H} be a set of graphs. A graph is called \mathcal{H} -*free* if it does not contain any graph of \mathcal{H} as an induced subgraph.

Observe that by the above discussion, the obstruction sets shown in the figure immediately provide polynomial-time recognition algorithms for split graphs, threshold graphs, cographs, and, trivially, claw-free graphs. In all of these cases, faster recognition algorithms have been obtained [77, 82, 154, 188].

The Strong Perfect Graph Theorem

Characterizations of graph classes by forbidden induced subgraphs are not only algorithmic tools, but also the subject of profound mathematics. The most striking example is probably the case of *perfect* graphs, a class which we define now. Recall that the chromatic number of a graph is the minimum number of colors in any of its vertex-colorings that has no monochromatic edges (and that the maximum size of any clique is a lower bound on the chromatic number).

PERFECT. A graph G is called *perfect*, if for every induced subgraph H of G , the chromatic number of H is equal to the maximum size of any clique in H .

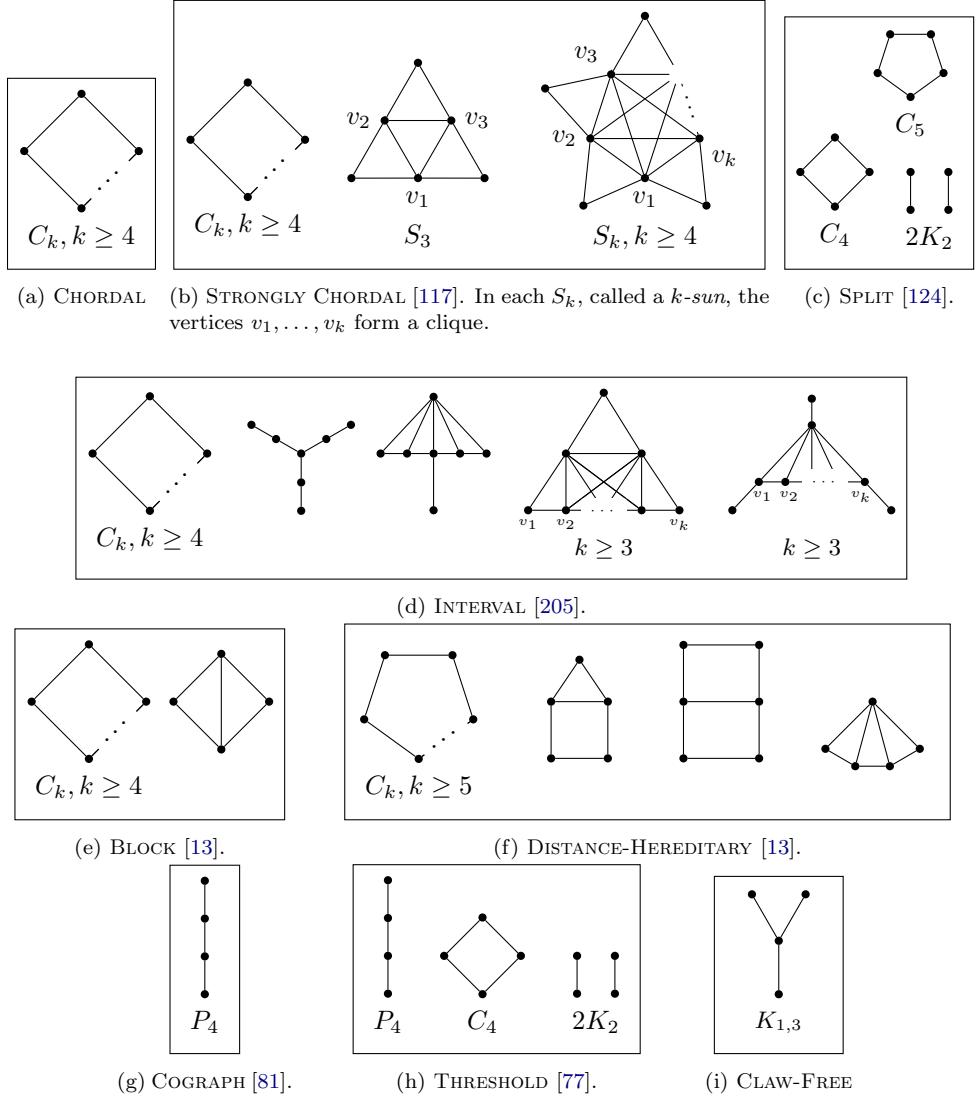


Figure 4.5: Obstruction sets for some of the graph classes discussed in this section.

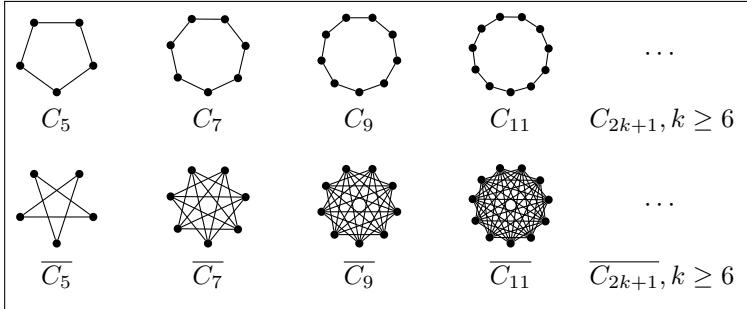


Figure 4.6: (The first few elements of) the obstruction set for perfect graphs. Note that C_5 and its complement are isomorphic, but for clarity we included two drawings.

In 1961, Claude Berge [18] conjectured that perfect graphs are precisely the graphs that neither have any cycle C_k with an odd number k of vertices nor its complement \overline{C}_k as an induced subgraph. We show these graphs in Figure 4.6. For every $k \geq 2$, the chromatic number of C_{2k+1} is three while its clique number is two, and the chromatic number of \overline{C}_{2k+1} is $k+1$, while its clique number is k . This conjecture has been called “probably the most beautiful open question in graph theory” [264, abstract], and nevertheless it remained open for over forty years. Its resolution by Chudnovsky, Robertson, Seymour, and Thomas [76] which was announced in 2002, was published in 2006.

4.7 Width Bounds

In this section, we consider upper and lower bound on the treewidth, clique-width, and mim-width of the previously introduced graph classes. We use the following notation. Let width be a width measure and CLS be a graph class. For $c \in \mathbb{N}$, we write

- $\text{width}(\text{CLS}) \leq c$ if for all $G \in \text{CLS}$, $\text{width}(G) \leq c$,
- $\text{width}(\text{CLS}) \geq c$ if for infinitely many $G \in \text{CLS}$, $\text{width}(G) \geq c$, and
- $\text{width}(\text{CLS}) = c$ if both $\text{width}(\text{CLS}) \leq c$ and $\text{width}(\text{CLS}) \geq c$.

For stronger lower bounds on the width of graph classes, we write

- $\text{width}(\text{CLS}) \geq \omega(1)$ if for *every* constant c , $\text{width}(\text{CLS}) \geq c$, and
- for a function $f: \mathbb{N} \rightarrow \mathbb{N}$, $\text{width}(\text{CLS}[n]) \geq f(n)$ if there are infinitely many values for n such that CLS contains an n -vertex graph G_n with $\text{width}(G_n) \geq f(n)$.

In some cases, we use the notation $\text{WIDTH } c$ to denote the class of all graphs G with $\text{width}(G) \leq c$. We present the bounds in increasing order of expressive strength of the width measures, and summarize all bounds discussed in this section at its end in Figure 4.9 (page 87).

4.7.1 Treewidth Bounds

Let T be a tree with at least two vertices. It is easy to observe that the treewidth of T is 1: To obtain a tree decomposition, we create a bag for each vertex $v \in V(T)$ containing only v , and for each edge $uv \in E(T)$, we create a bag containing only u and v . The tree of this decomposition can be obtained from T by subdividing each edge of its edges once. Each vertex created in a subdivision then gets assigned the corresponding ‘edge bag’.

On the other hand, a cycle has treewidth 2. Therefore, we have the following observation.

Observation 4.8. FOREST = TREEDIM 1.

Next, we consider cactus and outerplanar graphs. Observe that a cycle is a cactus and therefore outerplanar, so both graph classes contain infinitely many graphs of treewidth 2.

Theorem 4.9 (Bodlaender [27]; Arnborg and Proskurowski [8]).

$$\text{tw}(\text{CACTUS}) = \text{tw}(\text{OUTERPLANAR}) = 2.$$

Bodlaender [30] showed that the treewidth of each k -outerplanar graphs is at most $3k - 1$. Kammer and Tholey [175] later proved that this bound is tight, i.e. that there are infinitely many k -outerplanar graphs whose treewidth is indeed $3k - 1$.

Theorem 4.10 (Bodlaender [30], Kammer and Tholey [175]).

$$\text{tw}(k\text{-OUTERPLANAR}) = 3k - 1.$$

Before we proceed with the clique-width bounds on graph classes, let us observe that even the smallest classes among the ones we consider next have unbounded treewidth: they both contain all complete graphs.

Observation 4.11. $\text{tw}(\text{THRESHOLD}[n]) = \text{tw}(\text{BLOCK}[n]) = n - 1$.

4.7.2 Clique-Width Bounds

We first give some upper bounds on the clique-width of some graph classes. Recall that the complete graph on $n \geq 2$ vertices has clique-width 2. Therefore we have a trivial lower bound of 2 on the clique-width of several graph classes.

First, we consider cographs, and recall that this graph class includes threshold graphs. Courcelle and Olariu [89] showed that each cograph has clique-width at most 2; in fact they showed that a graph has clique-width 2 *if and only if* it is a cograph. One might therefore say that cographs are for clique-width what forests are for treewidth. (Recall that only edgeless graphs can have clique-width 1.)

Theorem 4.12 (Courcelle and Olariu [89]). COGRAPH = CLIQUE-WIDTH 2.

Golumbic and Rotics [148] showed that distance-hereditary graphs have clique-width at most 3. Now, each block graph is distance-hereditary, and each forest is a block graph. Consider P_n for $n \geq 4$, a path on at least four vertices, which is clearly a forest. As we have seen earlier (cf. Figure 4.5), cographs are precisely the graphs that do not contain P_4 as an induced subgraph. By Theorem 4.12, cographs are also precisely the graphs of clique-width at most 2, therefore P_n has clique-width three.

Theorem 4.13 (Golumbic and Rotics [148]).

$$\text{cw}(\text{FOREST}) = \text{cw}(\text{BLOCK}) = \text{cw}(\text{DISTANCE-HEREDITARY}) = 3.$$

We also consider several graph classes that do not completely avoid P_4 s, but that have ‘few P_4 s’, namely P_4 -sparse and P_4 -tidy graphs, and (q, t) -graphs. The following upper bounds on the clique-width of these graph classes are known due to Courcelle, Makowsky, and Rotics [88], and Makowsky and Rotics [214].

Theorem 4.14 ([88, 214]).

- For all $q \geq 4$, $\text{cw}((q, q - 4)\text{-GRAPH}) \leq q$.
- For $q \geq 7$, $\text{cw}((q, q - 3)\text{-GRAPH}) \leq q$.
- $\text{cw}(P_4\text{-TIDY}) \leq 4$, therefore $\text{cw}(P_4\text{-SPARSE}) \leq 4$.

Moreover, Vanherpe [278] showed that the class of *partner-limited* graphs, which generalize P_4 -tidy graphs, also has bounded clique-width.

Theorem 4.15 (Vanherpe [278]). $\text{cw}(\text{PARTNER-LIMITED}) \leq 4$.

Makowsky and Rotics also proved several lower bounds on the clique-width of such classes.

Theorem 4.16 (Makowsky and Rotics [214]).

- For $4 \leq q \leq 6$, $\text{cw}((q, q - 3)\text{-GRAPH}[n]) \geq \Omega(\sqrt{n})$.
- $\text{cw}((7, 5)\text{-GRAPH}[n]) \geq \Omega(\sqrt{n})$.
- For $q \geq 4$, $\text{cw}((q, q - 1)\text{-GRAPH}[n]) \geq \Omega(\sqrt{n}/q)$.

To the best of our knowledge, the clique-width of the following graph classes is still unknown. Settling this question which was posed by Makowsky and Rotics in 1999 would result in a complete dichotomy for the clique-width of (q, t) -graph classes for all admissible values of q and t .

Open Problem 4.1 ([214]). For constant $q \geq 8$, is the clique-width of $(q, q-2)$ -graphs bounded by a constant or not?

Next, we consider a set of graph classes that allows us to conclude that all graph classes for which we give upper bounds on the mim-width in the next section, have unbounded clique-width. The graph classes are: INTERVAL, BIPARTITE PERMUTATION, DILWORTH-2, and Co-2-DEGENERATE. As we see in Figure 4.9, each of the graph classes in the ‘bounded mim-width bubble’ contains one of these four classes.

First, we consider interval and permutation graphs; Golumbic and Rotics showed that there are graphs in both classes whose clique-width is at least the square-root of the number of their vertices. The graphs provided in the proof are obtained from an $n \times n$ -grid, and adding edges to obtain a graph in the targeted class. This is a common theme in obtaining lower bounds both on the clique-width and on the mim-width of graph classes.

Theorem 4.17 (Golumbic and Rotics [148]).

$$\text{cw}(\text{INTERVAL}[n]) \geq \sqrt{n} \text{ and } \text{cw}(\text{PERMUTATION}[n]) \geq \sqrt{n}.$$

Brandstädt and Lozin [53] strengthened the bound for permutation graphs to bipartite permutation graphs by a construction that follows a similar strategy.

Theorem 4.18 (Brandstädt and Lozin [53]).

$$\text{cw}(\text{BIPARTITE PERMUTATION}[n]) \geq \sqrt{n}/6.$$

Korpelainen et al. [193] studied the class of *split permutation* graphs, and showed that these coincide with the class of dilworth-2 graphs. Moreover, they gave a construction of countably infinite split permutation graphs with unbounded clique-width. Together with a theorem due to Courcelle [86] which asserts that each countably infinite graph of clique-width 2^{2k+1} has a finite induced subgraph of clique-width k , this gives the following result.

Theorem 4.19 (Korpelainen et al. [193]). $\text{cw}(\text{DILWORTH-2}) \geq \omega(1)$.

Courcelle and Olariu [89] showed that taking the complement of a graph does not increase its clique-width too much. Concretely, for any graph G with complement \overline{G} , $\text{cw}(\overline{G}) \leq 2 \cdot \text{cw}(G)$. Since the $(n \times n)$ -grid (defined below) has clique-width $n + 1$ whenever $n \geq 3$ [148], and each grid is 2-degenerate, this implies the following lower bound on the clique-width of co-2-degenerate graphs which was also observed by Vatshelle [281].

Theorem 4.20 ([89, 148, 281]). $\text{cw}(\text{Co-2-DEGENERATE}[n]) \geq \Omega(\sqrt{n})$.

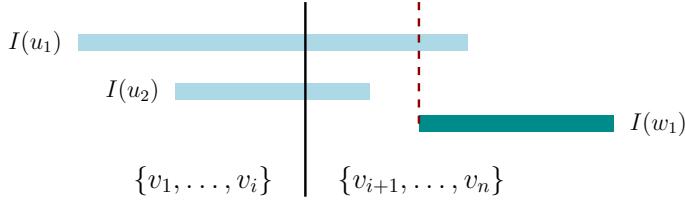


Figure 4.7: Illustration of the argument given in the proof of Theorem 4.21. On the left side of the cut we have the vertices whose interval has its left endpoint before $\ell(v_i)$ and on the right side the vertices whose interval has the left endpoint after $\ell(v_i)$.

4.7.3 Mim-Width Bounds

A large portion of the upper bound of the mim-width on several graph classes is due to the work of Belmonte and Vatshelle [17]. Most of these bounds are for intersection graph classes or classes that are defined in terms of some ordering on the vertices, and decompositions are typically found by tweaking an intersection model/vertex ordering into a branch decomposition of bounded mim-width. We illustrate how this is done for interval graphs in the proof of Theorem 4.21 below. More recently, several nontrivial lower bounds on the mim-width of graph classes have been shown, and we discuss them after the upper bounds.

Upper Bounds

We present the upper bounds in increasing order of mim-width value, and start at interval graphs.

Theorem 4.21 (Belmonte and Vatshelle [17]). $\text{Imimw}(\text{INTERVAL}) = 1$. Moreover, a linear branch decomposition witnessing this bound can be found in time $\mathcal{O}(n + m)$ for an n -vertex m -edge interval graph.

Proof. We can compute in time $\mathcal{O}(n + m)$ an interval model $\{I(v)\}_{v \in V(G)}$ of the input interval graph G where the endpoints of all intervals are pairwise distinct [48]. For each interval $I(v)$, we denote by $\ell(v)$ the left endpoint of I_v and by $r(v)$ its right endpoint, i.e. $I(v) = [\ell(v), r(v)]$. Let v_1, \dots, v_n be the ordering on $V(G)$ according to increasing value of the left endpoints, meaning $\ell(v_1) < \dots < \ell(v_n)$. We show that the mim-width of this ordering is at most 1.

Suppose the contrary, then for some $i \in [n - 1]$, there is an induced matching $\{u_1 w_1, u_2 w_2\}$ in $G[\{v_1, \dots, v_i\}, \{v_{i+1}, \dots, v_n\}]$. See Figure 4.7 for an illustration of the following argument. Suppose wlog that $\{u_1, u_2\} \subseteq \{v_1, \dots, v_i\}$ and $\{w_1, w_2\} \subseteq \{v_{i+1}, \dots, v_n\}$. By construction, and the choice of the cut, the left endpoints of $I(w_1)$ and $I(w_2)$ are to the right of the left endpoints of both $I(u_1)$ and $I(u_2)$. (When ‘projecting’ the cut to the interval model, these pairs of intervals start on opposite

sides.) Now, w_1 is adjacent to u_1 but not to u_2 , which implies that

$$r(u_2) < \ell(w_1) < r(u_1). \quad (4.1)$$

Similarly, since w_2 is adjacent to u_2 but not to u_1 , this implies that

$$r(u_1) < \ell(w_2) < r(u_2). \quad (4.2)$$

Putting Inequalities (4.1) and (4.2) together, we obtain that

$$r(u_2) < r(u_1) < r(u_2) < r(u_1),$$

which is clearly a contradiction. \square

Similar constructions show the following bounds.

Theorem 4.22 (Belmonte and Vatshelle [17]).

$$\text{lmimw(PERMUTATION)} = \text{lmimw(BICONVEX)} = \text{lmimw(CONVEX)} = 1.$$

In each of these classes, a linear branch decomposition witnessing the bound can be found in time $\mathcal{O}(n + m)$ for an n -vertex m -edge member of these graph classes.

In all bounds we have seen so far, the proofs have been constructive in the sense that we could always find a good branch decomposition in polynomial time. In the following result this is currently not the case. The bound for leaf power graphs relies on a given leaf root, and it is still unknown whether or not leaf powers can be recognized in polynomial time, see Section 5.3. For the subclass of rooted directed path graphs, a polynomial-time recognition algorithm is known, and we can provide a branch decomposition of bounded mim-width.

As opposed to the previous upper bounds, in both cases the *linear* mim-width is unbounded, since both classes contain trees which are of unbounded linear mim-width [161].

Theorem 4.23. *Let $\text{CLS} \in \{\text{LEAF POWER}, \text{ROOTED DIRECTED PATH}\}$. Then,*

$$\text{mimw}(\text{CLS}) = 1 \text{ and } \text{lmimw}(\text{CLS}[n]) \geq \Omega(\log n).$$

Moreover, given an n -vertex m -edge rooted directed path graph, we can construct a branch decomposition witnessing the upper bound in time $\mathcal{O}(n + m)$.

Proof. We argue the upper bound on the mim-width, and note that the lower bound on the linear mim-width follows from the fact that both graph classes contain all trees: it is known that $\text{lmimw}(\text{TREE}[n]) \geq \Omega(\log n)$ [161]. The mim-width upper bound for leaf power graphs is shown in Section 5.3, specifically in Corollary 5.10.

We now prove the mim-width upper bound for rooted directed path graphs. Let G be a rooted directed path graph on n vertices and m edges. Using the algorithm

due to Gavril [138] we can find in time $\mathcal{O}(n + m)$ a rooted directed tree T and a collection $\{P(v)\}_{v \in V(G)}$ of directed paths in T that form an intersection model of G . Note that we may assume that for each leaf of T , there is at least one path $P(v)$ (for some $v \in V(G)$) ending in that leaf. Otherwise, we may simply remove it. We obtain a branch decomposition (T', \mathcal{L}) from T in the following steps.

1. For each node $t \in V(T)$ with degree more than three, we replace t by a full binary tree (and adjust the paths $\{P(v)\}_{v \in V(G)}$ accordingly). If the root has degree more than two, we do the same. Denote the resulting tree by T'' and the paths by $\{P''(v)\}_{v \in V(G)}$.
2. For each vertex $v \in V(G)$, consider the corresponding path $P''(v)$ in T'' and do the following: Let t_v be the last (i.e. lowest) vertex of $P''(v)$. If t_v is a leaf, then we add a new child x_v of t_v to T'' . If t_v is not a leaf, then let s be an arbitrary child of t_v . We subdivide the edge $t_v s$ and obtain a subdivision vertex y_v , to which we add a child x_v .
3. We let T' be the tree obtained from T'' by performing the previous step for all vertices in G . We let $\mathcal{L}: V(G) \rightarrow L(T')$ be such that $\mathcal{L}(v) = x_v$ for all $v \in V(G)$, where x_v is the leaf introduced for v in the previous step.

It is not difficult to verify that (T', \mathcal{L}) is a rooted branch decomposition of G , and that the above process takes time $\mathcal{O}(n + m)$. We show that it has mim-width at most 1. (The argument is very similar to the one for interval graphs.)

Let $t \in V(T')$ be any non-root node, and let p be the parent of t in T . Suppose for a contradiction that $G[V_t, \bar{V}_t]$ has an induced matching of size two, and denote it by $\{u_1 v_1, u_2 v_2\}$, where $\{u_1, u_2\} \subseteq V_t$ and $\{v_1, v_2\} \subseteq \bar{V}_t$. Let $t^* \in V(T)$ be the nearest node above t that is also contained in T'' . Since both u_1 and u_2 have a neighbor in \bar{V}_t , and by the construction given above, we know that $P''(u_1)$ and $P''(u_2)$ both contain the node t^* . For all $i \in [2]$, let P_i^* be the subpath of $P''(u_i)$ ending in t^* . Since T'' is rooted, and all paths $P''(\cdot)$ are directed, we know that at least one of $P_1^* \subseteq P_2^*$ and $P_2^* \subseteq P_1^*$ is true. Now, v_1 is a vertex in \bar{V}_t that is adjacent to u_1 but not to u_2 . This means that $P_2^* \subset P_1^*$. On the other hand, v_2 is a vertex that is adjacent to u_2 but not to u_1 , therefore $P_1^* \subset P_2^*$. We obtained that $P_1^* \subset P_2^* \subset P_1^* \subset P_2^*$, a contradiction. \square

Next, we move on to graph classes of linear mim-width at most 2.

Theorem 4.24 (Belmonte and Vatshelle [17]). *Let CLS be one of the following graph classes: CIRCULAR ARC, CIRCULAR PERMUTATION, BOUNDED TOLERANCE, TRAPEZOID. Then,*

$$\text{limimw}(\text{CLS}) \leq 2.$$

Moreover, a linear branch decomposition witnessing the bound can be found in time $\mathcal{O}(n^2)$ for n -vertex bounded tolerance or trapezoid graphs, and in time $\mathcal{O}(n + m)$ for n -vertex m -edge circular arc or circular permutation graphs.

Regarding the previous theorem, recall that recognizing BOUNDED TOLERANCE graphs is NP-complete [222]. However, since each bounded tolerance graph is also a trapezoid graph [41], we can construct a linear branch decomposition of a bounded tolerance graph using the recognition algorithm for trapezoid graphs, and then following the construction for trapezoid graphs [17].

Next, we move on to graph classes that have a parameter k in their definition; they come in three groups, one for which the upper bound on the linear mim-width is at most k , one for which it is at most $k+1$, and one where the bound is at most $2k$.

Theorem 4.25 (Belmonte and Vathselle [17]). *Let CLS_k be one of the following graph classes: DILWORTH- k for $k \geq 1$ or k -TRAPEZOID for $k \geq 3$. Then,*

$$\text{Imimw}(\text{CLS}_k) \leq k.$$

Moreover, for dilworth- k graphs on n vertices, a linear branch decomposition witnessing the bound can be found in time $\mathcal{O}(k^2 \cdot n^2)$, and for k -trapezoid graphs, a linear branch decomposition can be found in polynomial time given a k -trapezoid model.

The following bound for complements of k -degenerate graphs is not explicitly proved in [17], but can be observed via the following argument [282]. Suppose G is co- k -degenerate and consider a cut (A, \bar{A}) in the corresponding linear order on $V(G)$. Each vertex $v \in A$ has at most k non-neighbors in \bar{A} , meaning that if v is the endpoint of an edge of an induced matching in $G[A, \bar{A}]$, there can be at most k other edges, otherwise it is not an induced matching.

Theorem 4.26 ([17]). $\text{Imimw}(\text{Co-}k\text{-DEGENERATE}) \leq k+1$, and a linear branch decomposition of an n -vertex m -edge co- k -degenerate graph witnessing this bound can be found in time $\mathcal{O}(n+m)$.

Theorem 4.27 ([17], Fomin et al. [127]). *Let CLS_k be one of the following graph classes: k -TRAPEZOID for $k \geq 3$, CIRCULAR k -TRAPEZOID for $k \geq 2$, k -POLYGON for $k \geq 1$, H -GRAPH where $k = |E(H)|$. Then,*

$$\text{Imimw}(\text{CLS}_k) \leq 2k,$$

and given an n -vertex graph together with an intersection model, a linear branch decomposition witnessing the bound can be found in polynomial time. Moreover,

- for H -graphs when H is a tree, a linear branch decomposition witnessing the bound can be found in time $\mathcal{O}(n^{|E(H)|^2})$ even if an intersection model is not provided, and
- for k -polygon graphs on n vertices, a linear branch decomposition witnessing the bound can be found in time $\mathcal{O}(4^k \cdot n^2)$ even if an intersection model is not provided.

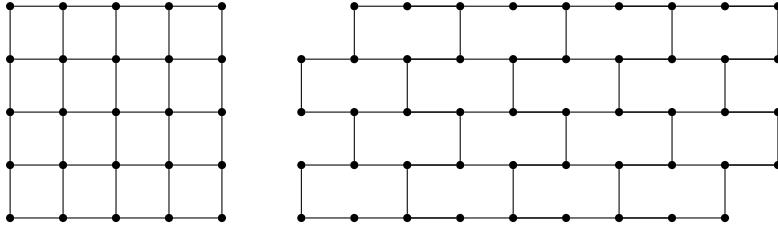


Figure 4.8: A 5×5 -grid (grid_5) on the left and an elementary 5×5 -wall (wall_5) on the right.

Lower Bounds

Before we list lower bounds on the mim-width of graph classes in general, we discuss the mim-width of some special types of graphs. The archetypical example of graphs of unbounded width are *square grids*. For $n \in \mathbb{N}$, the $n \times n$ -grid is a graph grid_n whose vertex set can be identified with $[n] \times [n]$ in such a way that $\{(i, j), (i', j')\} \in E(\text{grid}_n)$ if and only if $|(i - i') + (j - j')| = 1$. We illustrate such a graph in Figure 4.8.

Jelínek [174] showed that in each branch decomposition of grid_n , there is a cut (A, \overline{A}) such that $G[A, \overline{A}]$ contains a matching of size at least $n - 1$. Vatshelle [281] proved that if $G[A, \overline{A}]$ is d -degenerate and has a matching of size μ , then $G[A, \overline{A}]$ has an *induced* matching of size $\frac{\mu}{d+1}$. Since grid graphs are 2-degenerate, this shows that $\text{mimw}(\text{grid}_n) \geq \frac{n-1}{3}$. Now, it is not difficult to observe that grid graphs are planar bipartite graphs: in Figure 4.8 we show a planar drawing of a grid graph; to obtain a 2-partition (A, B) of the vertices of a grid into two independent sets, traverse the diagonals starting in the top left and moving towards the bottom right, and alternate in adding the vertices to A and to B .

Theorem 4.28 (Jelínek [174] and Vatshelle [281]). $\text{mimw}(\text{grid}_n) \geq \frac{n-1}{3}$. Therefore, $\text{mimw}(\text{PLANAR BIPARTITE}[n]) \geq \Omega(\sqrt{n})$.

Kang et al. [176] proved a lower bound of $\sqrt{\log(n/2)}$ on the mim-width of split graphs. Mengel [220] observed the following elegant trick based on grid graphs that improves this lower bound to roughly $\frac{\sqrt{n}}{6}$. Consider a bipartite graph G with bipartition (X, Y) and let (A, \overline{A}) be a cut in a branch decomposition of G , and M an induced matching in $G[A, \overline{A}]$. Since G is bipartite, M necessarily contains a matching M' of size $|M'|/2$ such that $V(M') \cap A \subseteq X$ and $V(M') \cap \overline{A} \subseteq Y$ (up to renaming). Therefore, if we add any number of edges between vertices in X or in Y , M' still remains an induced matching. Now, take the grid graph grid_n with bipartition (X, Y) , make X a clique and call the resulting graph H . By the previous discussion and Theorem 4.28, $\text{mimw}(H) \geq \frac{n-1}{6}$. Moreover H is a (strongly chordal) split graph. With more refined techniques, and using a construction due to Brault-Baron et al. [54], Mengel showed the following lower bounds. The lower bound for co-comparability graphs improves upon an $\Omega(\sqrt{n})$ lower bound due to Kang et al. [176].

Theorem 4.29 (Mengel [220]). *Let CLS be one of STRONGLY CHORDAL SPLIT, CIRCLE, and CO-COMPARABILITY. Then, $\text{mimw}(\text{CLS}[n]) \geq \Omega(n)$.*

As mentioned above, the constructions due to Mengel utilize the following lower bound construction due to Brault-Baron et al. [54]. (A graph is chordal bipartite if it is bipartite and each cycle of length at least six has a chord.)

Theorem 4.30 (Brault-Baron et al. [54]).

$$\text{mimw}(\text{CHORDAL BIPARTITE}[n]) \geq \Omega(n).$$

An immediate consequence of this lower bound together with the above described trick due to Mengel gives the following bound on the mim-width of co-bipartite graphs.

Corollary 4.31 ([54, 220]). $\text{mimw}(\text{CO-BIPARTITE}[n]) \geq \Omega(n)$.

We finish this section by giving lower bounds on some other basic types of graphs that were recently observed by Brettel et al. [55]. Specifically, we consider (*elementary*) *walls*. These graphs turn out to be useful as well in proving lower bound on the mim-width of graph classes as well. Rather than formally defining walls, we illustrate these graphs in Figure 4.8. We denote the elementary $n \times n$ -wall by wall_n .

Theorem 4.32 (Brettel et al. [55]). $\text{mimw}(\text{wall}_n) \geq \frac{\sqrt{n}}{30}$.

4.7.4 Sim-Width

As mentioned in the beginning of this chapter, chordal graphs are a very central graph class in algorithmic graph theory. Unfortunately, as we have just seen, chordal graphs have unbounded mim-width. In the search for a width parameter that is bounded on chordal graphs, Kang et al. [176] introduced *special induced matching* width (sim-width for short). Sim-width is also based on branch decompositions over the vertex set of a graph, but the value of the cut is computed as the maximum size of what is called a *special induced matching*. For a graph G and a set $A \subseteq V(G)$, a special induced matching in $G[A, \bar{A}]$ is an induced matching M such that additionally, $E(G[V(M)]) = M$, meaning that there are no edges between the endpoints of M even *inside* A and \bar{A} . Throughout, we denote the sim-width of a graph G by $\text{simw}(G)$.

Theorem 4.33 (Kang et al. [176]).

$$\text{simw}(\text{CHORDAL}) = \text{simw}(\text{CO-COMPARABILITY}) = 1.$$

Moreover, given an n -vertex m -edge graph of either graph class, we can compute a branch decomposition witnessing the bound in time $\mathcal{O}(n + m)$.

On the negative side, Kang et al. also proved that the class of circle graphs has unbounded sim-width.

Theorem 4.34 (Kang et al. [176]). $\text{simw}(\text{CIRCLE}) \geq \omega(1)$.

The previous lower bound is proved via a lemma that shows that if a graph has large mim-width and contains no K_3 as an induced subgraph, then it has large sim-width. We can apply the same reasoning to grid graphs, which shows that grids have unbounded sim-width. The resulting observation completes our picture on width bounds of the graph classes we considered here.

Observation 4.35. $\text{simw}(\text{PLANAR BIPARTITE}) \geq \omega(1)$.

So far, there is no problem that is known to have an XP-time algorithm parameterized by the sim-width of a given branch decomposition. We cannot expect that all mim-width based algorithmic results can be lifted to sim-width: All (σ, ρ) -domination problems (for finite or co-finite σ and ρ) are known to be XP parameterized by mim-width (given a decomposition), and there are several such problems that are NP-complete on chordal graphs [143, 194].

Nevertheless, we have seen that MAXIMUM INDEPENDENT SET is polynomial-time solvable on chordal graphs, and it would be interesting to see if this can be lifted to graphs of bounded sim-width, or at least to graphs of sim-width 1.

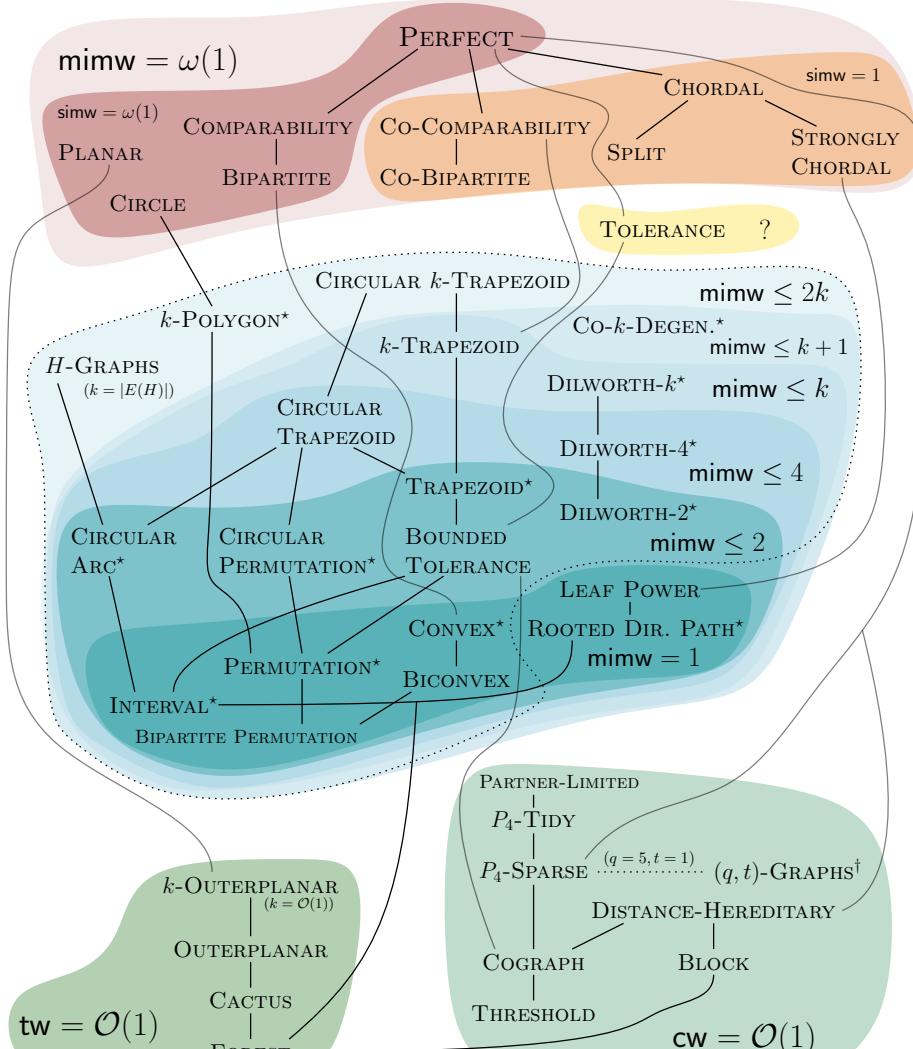
Open Problem 4.2. Is there a polynomial-time algorithm for MAXIMUM INDEPENDENT SET when the input graph is given together with one of its branch decompositions of sim-width 1?

4.7.5 Bounds for \mathcal{H} -Free Graphs

Another line of research is to determine upper and lower bounds of width parameters of graph classes defined by some set \mathcal{H} of forbidden induced subgraphs, typically for small sets \mathcal{H} . For a single graph H , it is known that the class H -FREE has bounded clique-width if and only if H is an induced subgraph of P_4 , see for instance [95] for a proof. Brettel et al. [55] showed that the same dichotomy is true with respect to mim-width. For treewidth, we cannot obtain any nontrivial result for one forbidden induced subgraph. For H -free graphs to have bounded treewidth, H must also be an induced subgraph of P_4 by the above mentioned result regarding clique-width. P_3 -free graphs still contain all complete graphs which therefore have unbounded treewidth, and forbidding K_1 or K_2 results in a graph without vertices, and edges, respectively.

For two forbidden induced subgraphs, Bodlaender et al. [31] showed a complete dichotomy for the boundedness of treewidth, while Dabrowski and Paulusma [95] come close to a dichotomy for boundedness of clique-width, and Brettel et al. [55] come close to a dichotomy for boundedness of mim-width, see also [56]. Motivated by the fact that a graph class often admits a polynomial-time algorithm for a problem when its *atoms*³ have bounded clique-width, Dabrowski et al. [97] recently studied

³A graph is an *atom* if it has no clique-separator, i.e. no separator that is a clique.



[†] $q, t = \mathcal{O}(1)$; $q \geq 4, t \geq 0$, and either
 $q \leq 6$ and $t \leq q-4$, or $q \geq 7$ and $t \leq q-3$

Figure 4.9: The width bounds discussed in this section. For two graph classes that are connected by an edge, the graph class that is higher up contains the one below. All bounded mim-width graph classes except the ones containing forests have bounded linear mim-width. In graph classes marked with ‘*’, we know how to efficiently compute bounded mim-width branch decompositions.

the clique-width of H -free and (H_1, H_2) -free atoms. The clique-width of \mathcal{H} -free graph classes is a deeply studied topic, and we refer to the recent survey of Dabrowski et al. [96].

Kang et al. [176] also proved that several H -free subclasses of chordal and co-comparability graphs have bounded mim-width: For $t \in \mathbb{N}$, the graph $K_t \boxminus K_t$ is the graph obtained from two t -cliques by adding a perfect matching between them, and $K_t \boxminus S_t$ is the graph obtained from a t -clique and an independent set of size t by adding a perfect matching between them. They first showed that interval graphs are $(K_3 \boxminus S_3)$ -free chordal graphs, and that the mim-width of $(K_t \boxminus S_t)$ -free chordal graphs is at most $t - 1$. Then they showed that $(K_3 \boxminus K_3)$ -free co-comparability graphs are permutation graphs, and that $(K_t \boxminus K_t)$ -free co-comparability graphs have mim-width at most $t - 1$.

5

Mim-Width of Graph Powers

In this chapter, we show that taking an (arbitrarily high) power of a graph of bounded mim-width increases its mim-width by a factor of at most two. We give more refined bounds on the mim-width of powers of graphs of bounded treewidth and clique-width. The former allows us to conclude that leaf power graphs have mim-width 1.

Before we proceed with the mim-width bounds, we describe the behaviour of the parameters treewidth and clique-width under the operation of taking powers.

First, we quickly observe we cannot give any nontrivial bound on the treewidth of powers of graphs of bounded treewidth. Take the star S_n on n vertices. Since S_n has diameter 2, we have that $S_n^2 = K_n$. Therefore, $\text{tw}(S_n) = 1$ and $\text{tw}(S_n^2) = n - 1$.

Observation 5.1. For each $n \in \mathbb{N}$, there is a graph G_n on n vertices such that $\text{tw}(G_n) = 1$ and $\text{tw}(G_n^2) = n - 1$.

Clearly, this construction fails to rule out clique-width bounds on the power of graphs of bounded clique-width; as we have seen in Section 3.3, $\text{cw}(K_n) = 2$. Todinca indeed showed that for fixed r , taking the r -th power of a graph of bounded clique-width w results in a graph of bounded clique-width, albeit (possibly) with an exponential increase of the bound.

Theorem 5.2 (Todinca [276]). *Let G be a graph with $\text{cw}(G) \leq w$, and r be a positive integer. Then, $\text{cw}(G^r) \leq 2wr^w$.*

Recall that by Theorem 3.19, the clique-width of the class of graphs of treewidth w is $2^{\Theta(w)}$. It is therefore interesting that Gurski and Wanke [151] showed that for constant r , the class of r -powers of graphs of treewidth w has clique-width $2^{\mathcal{O}(w)}$, rather than the trivial bound of $2^{2^{\mathcal{O}(w)}}$ that follows immediately from a combination of Theorems 3.19 and 5.2. The precise bound is as follows.

Theorem 5.3 (Gurski and Wanke [151]). *Let G be a graph with $\text{tw}(G) \leq w$, and r be a positive integer. Then, $\text{cw}(G^r) \leq 2(r + 1)^w + 1 - 2$.*

5.1 General Graphs

We now show that taking the power of any graph does not increase its mim-width by more than a factor of two.

Theorem 5.4. *Let r be a positive integer and let G be a graph. Then, $\text{mimw}(G^r) \leq 2 \cdot \text{mimw}(G)$. Moreover, any branch decomposition of G of mim-width w has mim-width at most $2w$ for G^r .*

Proof. Assume that there is a branch decomposition of mim-width w for the graph G . We show that the same branch decomposition has mim-width at most $2w$ for G^r .

We consider a cut (A, \overline{A}) associated with some edge of the branch decomposition. Let M be a maximum induced matching across the cut for G^r . To prove our claim, it suffices to construct an induced matching M' in G across the cut (A, \overline{A}) such that $|M'| \geq \frac{|M|}{2}$.

We begin by noticing that for an edge $uv \in M$, the distance between u and v is at most r in G . For each such edge $uv \in M$, we let P_{uv} denote some shortest path between u and v in G (including the endpoints u and v).

Claim 5.4.1. *Let $uv, wx \in M$ be two distinct edges in M . Then P_{uv} and P_{wx} are vertex disjoint.*

Proof. We may assume that $u, w \in A$ and $v, x \in \overline{A}$. Now assume for the sake of contradiction there exists a vertex $y \in V(P_{uv}) \cap V(P_{wx})$. Because both paths have length at most r , we have that $\text{dist}_G(u, y) + \text{dist}_G(y, v) \leq r$, and $\text{dist}_G(w, y) + \text{dist}_G(y, x) \leq r$. Adding these together, we get

$$\text{dist}_G(u, y) + \text{dist}_G(y, v) + \text{dist}_G(w, y) + \text{dist}_G(y, x) \leq 2r. \quad (5.1)$$

Since uv and wx are both in M , there cannot exist edges ux and wv in G^r . Hence, their distance in G is strictly greater than r , i.e.

$$\text{dist}_G(u, y) + \text{dist}_G(y, x) \geq \text{dist}_G(u, x) > r, \text{ and } \text{dist}_G(w, y) + \text{dist}_G(y, v) > r.$$

Putting these together, we obtain a contradiction with Equation (5.1):

$$\text{dist}_G(u, y) + \text{dist}_G(y, x) + \text{dist}_G(w, y) + \text{dist}_G(y, v) > 2r$$

This concludes the proof of the claim. \square

Because for each $uv \in M$, one endpoint of P_{uv} lies in A and the other one lies in \overline{A} , there must exist at least one point at which the path crosses from A to \overline{A} . For each $uv \in M$ with $u \in A$ and $v \in \overline{A}$, we let $u'v' \in E(P_{uv})$ denote an edge in G such that $u' \in A$ and $v' \in \overline{A}$.

We plan to construct our matching M' by picking a subset of such edges. However, we cannot simply take all of them, since some pairs may be incompatible in the sense

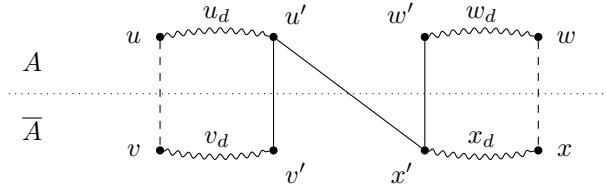


Figure 5.1: Structure of two paths P_{uv} and P_{wx} when the edge $u'x'$ exists in G . Dashed edges appear in G^r , solid edges appear in G , squiggly lines are (shortest) paths existing in G (possibly of length 0, and possibly crossing back and forth across the cut).

that they will not form an induced matching across the cut (A, \bar{A}) . We examine the structures that arise when two such edges $u'v'$ and $w'x'$ are incompatible, and cannot both be included in the same induced matching across the cut. For easier readability, we let α_d be a shorthand notation for $\text{dist}_G(\alpha, \alpha')$ for $\alpha \in \{u, v, w, x\}$.

The setting of the following claim is illustrated in Figure 5.1.

Claim 5.4.2. *Let $uv, wx \in M$ with $\{u, w\} \subseteq A$ and $\{v, x\} \subseteq \bar{A}$ be two distinct edges of M and let $u'v'$ and $w'x'$ be edges on the shortest paths as defined above. If there is an edge $u'x' \in E(G)$, then the following hold.*

- (a) $u_d + x_d = r$
- (b) $u_d + v_d = w_d + x_d = r - 1$
- (c) $w_d = u_d - 1$

Proof. (a). Since ux is not an edge in G^r , the distance between u and x must be at least $r+1$ in G , and so $u_d + x_d$ must be at least r . It remains to show that $u_d + x_d \leq r$ for equality to hold. Similarly to the proof of Claim 5.4.1, we know that P_{uv} and P_{wx} both are of length at most r . We get

$$u_d + v_d + w_d + x_d \leq 2r - 2. \quad (5.2)$$

The ‘ -2 ’ at the end is because we do not include the length contributed by edges $u'v'$ and $w'x'$ in our sum. Now assume for the sake of contradiction that $u_d + x_d \geq r+1$. Then we get that

$$v_d + w_d \leq 2r - 2 - r - 1 = r - 3.$$

Because $\text{dist}_G(v', w') \leq 3$ (follow the edges $u'v' \rightarrow u'x' \rightarrow w'x'$), this implies that $\text{dist}_G(v, w) \leq r$, and the edge vw would hence exist in G^r . This contradicts that uv and wx were both in the same induced matching M .

(b). Assume for the sake of contradiction that $u_d + v_d \leq r - 2$. Then, rather than Equation (5.2), we get the following bound:

$$u_d + v_d + w_d + x_d \leq 2r - 3$$

By (a) we know that $u_d + x_d = r$, so by a similar argument as above we get that $v_d + w_d \leq r - 3$, obtaining a contradiction. An analogous argument holds for $w_d + x_d$.

(c). This follows immediately by substituting (a) into (b). \square

We will now construct our induced matching M' . (Recall for the following arguments that $u \in A$ and $v \notin A$.) We construct two candidates for M' , and we will pick the biggest one. First, we construct M'_0 by including $u'v'$ for each edge $uv \in M$ where $\text{dist}_G(u, u')$ is even. Symmetrically, M'_1 is constructed by including $u'v'$ if $\text{dist}_G(u, u')$ is odd. Clearly, at least one of M'_0 and M'_1 contains at least $\frac{|M|}{2}$ edges. It remains to show that M' indeed forms an induced matching across the cut (A, \bar{A}) in G .

Consider two distinct edges $u'v'$ and $w'x'$ from M' . By Claim 5.4.1, the corresponding paths P_{uv} and P_{wx} are vertex disjoint. If there is an edge violating that $u'v'$ and $w'x'$ are both in the same induced matching, it must be either $u'x'$ or $v'w'$. Without loss of generality we may assume it is an edge of the type $u'x'$. By Claim 5.4.2(c), we then have that the parities of $\text{dist}_G(u, u')$ and $\text{dist}_G(w, w')$ are different. However, by the construction of M' , this is not possible. This concludes the proof. \square

5.2 Graphs of Bounded Treewidth or Clique-Width

In [281, Section 4.2], it is shown that any graph of clique-width w or treewidth $w - 1$ has mim-width at most w . Theorem 5.4 hence implies that any power of a graph of clique-width w or treewidth $w - 1$ has mim-width at most $2w$. In this section we give tighter bounds on the mim-width of powers of graphs of bounded clique-width and powers of graphs of bounded treewidth. This will enable us to conclude that leaf power graphs have mim-width 1.

In particular, we show that r -th powers of graphs of treewidth $w - 1$, pathwidth w , or clique-width w all have mim-width at most w . We begin by proving the bound for graphs of bounded treewidth with the following lemma capturing the essential property used in the proof.

Lemma 5.5. *Let r and w be positive integers and let (A, B, C) be a vertex partition of graph G such that there are no edges between A and C and B has size w . Let $H := G^r$. Then, $\text{mim}_H(A \cup B) \leq w$.*

Proof. Let $B := \{b_1, b_2, \dots, b_w\}$. It is clear that for $v \in A \cup B$ and $z \in C$, $\text{dist}_G(v, z) \leq r$ if and only if there exists $i \in \{1, 2, \dots, w\}$ such that $\text{dist}_G(v, b_i) + \text{dist}_G(z, b_i) \leq r$.

Suppose for a contradiction that $\text{mim}_H(A \cup B) > w$. Let $\{y_1z_1, y_2z_2, \dots, y_tz_t\}$ be an induced matching of size $t \geq w + 1$ in $H[A \cup B, C]$. There are distinct integers $t_1, t_2 \in \{1, 2, \dots, t\}$ and an integer $j \in \{1, 2, \dots, w\}$ such that

$$\text{dist}_G(y_{t_1}, b_j) + \text{dist}_G(z_{t_1}, b_j) \leq r \text{ and } \text{dist}_G(y_{t_2}, b_j) + \text{dist}_G(z_{t_2}, b_j) \leq r.$$

Then we have either $\text{dist}_G(y_{t_1}, b_j) + \text{dist}_G(z_{t_2}, b_j) \leq r$ or $\text{dist}_G(y_{t_2}, b_j) + \text{dist}_G(z_{t_1}, b_j) \leq r$, which contradicts the assumption that $y_{t_1}z_{t_2}$ and $y_{t_2}z_{t_1}$ are not edges in H . We conclude that $\text{mim}_H(A \cup B) \leq w$. \square

Theorem 5.6. *Let r and w be positive integers and G be a graph that admits a nice tree decomposition of width w , all of whose join bags are of size at most w . Then the r -th power of G has mim-width at most w . Furthermore, given such a nice tree decomposition, we can output a branch decomposition of mim-width at most w in polynomial time.*

Proof. Let $H := G^r$, and let $(T, \mathcal{B} = \{B_t\}_{t \in V(T)})$ be a nice tree decomposition of G of width w , all of whose join bags have size at most w , with root node τ . We may assume that $B_\tau = \emptyset$ and subsequently that τ is a forget or a join node. (Otherwise, we add a path of forget nodes on top of τ and make the last node the new root of T .)

We obtain a branch decomposition (T', \mathcal{L}) as follows:

- Let T'' be the tree obtained from T by, for each forget node $t \in V(T)$ forgetting a vertex v , attaching a leaf node ℓ_v to t , and assigning $\mathcal{L}(v) := \ell_v$.
- We obtain T' from T'' by deleting degree 1 nodes that are not assigned by \mathcal{L} .

Since τ is either a forget node or a join node in (T, \mathcal{B}) , τ has not been removed in the second step, so $\tau \in V(T')$, and τ has degree 2 in T' . Furthermore, since for each vertex $v \in V(G)$, there is a unique forget node forgetting v in (T, \mathcal{B}) , and all leaves that are not assigned by \mathcal{L} have been removed, the map \mathcal{L} constructed above is a bijection. Thus, (T', \mathcal{L}) is a (rooted) branch decomposition with root node τ .

We consider a cut $(V_t, \overline{V_t})$ for some node $t \in V(T')$ in the rooted branch decomposition. If t is a leaf node, then $\text{mim}_H(V_t) \leq 1$. Assume t is an internal node, then t also appears in (T, \mathcal{B}) . Note that V_t consists precisely of all vertices that have been forgotten below t in (T, \mathcal{B}) . We argue that we can find a set of at most w vertices $S \subseteq \overline{V_t}$ such that S separates V_t from $\overline{V_t} \setminus S$ which by Lemma 5.5 will yield the claim.

Let $S := N(V_t) \cap \overline{V_t}$ and consider a vertex $x \in S$. By definition, x is a neighbor of some vertex y that has been forgotten below t . By property T2 of the definition of a tree decomposition (page 16) there is a node, say t' , such that $B_{t'}$ contains both x and y . Since y has been forgotten below t , t' is below t . As $x \in \overline{V_t}$, there is also a node above t , say t'' , such that $B_{t''}$ contains x (e.g. the bag below the one that forgets x). Since t lies on the path between t' and t'' in T , we have by property T3 of the definition of a tree decomposition that $x \in B_t$. We have that $S \subseteq B_t$.

Furthermore, S separates V_t from $\overline{V_t} \setminus S$. If t is a forget node, then by definition $|B_t| \leq w$ and hence $|S| \leq w$. If t is a join node, then by assumption $|B_t| \leq w$ and hence $|S| \leq w$. If t is an introduce node introducing a vertex $u \in V(G)$, then u cannot have any neighbor in V_t , since all vertices in V_t have been forgotten below t . Hence, $S \subseteq B_t \setminus \{u\}$ and we can conclude that $|S| \leq w$. \square

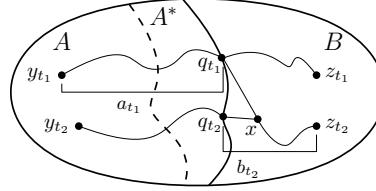


Figure 5.2: Illustration of the situation in the proof of Lemma 5.8.

By Lemma 3.3, any tree decomposition can be transformed in polynomial time into a nice tree decomposition of the same width. As in a tree decomposition of width $w - 1$, all bags (in particular bags at join nodes) have size at most w , the previous theorem implies the following. Note that the bound for pathwidth is slightly tighter than the one for treewidth, as in a path decomposition there are no join nodes.

Corollary 5.7. *Let r and w be positive integers and let G be a graph of treewidth $w - 1$ (pathwidth w). Then the r -th power of G has mim-width at most w and given a tree decomposition (path decomposition) of G of width $w - 1$ (w), one can compute a branch decomposition of mim-width w in polynomial time.*

Next, we consider powers of graphs of bounded clique-width. Recall that a graph is w -labeled if there is a labeling function $f : V(G) \rightarrow [w]$, and we call $f(v)$ the *label* of v . For a w -labeled graph G , we call the set of all vertices with label i the *label class i* of G . The following can be thought as a generalization of Lemma 5.5.

Lemma 5.8. *Let r and w be positive integers and let (A, B) be a vertex partition of graph G such that $G[A]$ is w -labeled and for every pair of vertices x, y in the same label class of $G[A]$, x and y have the same neighborhood in B . Let $H := G^r$. Then, $\text{mim}_H(A) \leq w$.*

Proof. Suppose for contradiction that $\text{mim}_H(A) > w$. Let $\{y_1 z_1, y_2 z_2, \dots, y_t z_t\}$ be an induced matching of size at least $w + 1$ in $H[A, B]$. For $i \in \{1, 2, \dots, t\}$, there is a path P_i of length at most r from y_i to z_i in G . Let $A^* \subseteq A$ be the set of vertices in A that have a neighbor in B . Let Q_i be the subpath of P_i from the last vertex in A^* to z_i , and q_i be the endpoint of Q_i different from z_i , and let R_i be the subpath of P_i from y_i to q_i . Let a_i be the length of R_i and b_i be the length of Q_i . By construction, $a_i + b_i \leq r$.

(For an illustration of the following argument, see Figure 5.2.) Since $t \geq w + 1$, there are two integers $t_1, t_2 \in \{1, 2, \dots, t\}$ such that q_{t_1} and q_{t_2} are contained in the same label class of $G[A]$. Since $a_{t_1} + b_{t_1} \leq r$ and $a_{t_2} + b_{t_2} \leq r$, we either have that $a_{t_1} + b_{t_2} \leq r$ or $a_{t_2} + b_{t_1} \leq r$.

Assume $a_{t_1} + b_{t_2} \leq r$. In this case, we show that the distance from y_{t_1} to z_{t_2} in G is at most r , which contradicts the assumption that $y_{t_1} z_{t_2}$ is not an edge of H . Note that two vertices in a label class of $G[A]$ have the same neighborhood in

B. Let x denote the vertex on Q_{t_2} that is adjacent to q_{t_2} . Then, as q_{t_1} and q_{t_2} are in the same label class of $G[A]$, we have that q_{t_1} is also adjacent to x . Therefore, $G[V(R_{t_1}) \cup (V(Q_{t_2}) \setminus \{q_{t_2}\})]$ contains a path of length at most r from y_{t_1} to z_{t_2} . Analogously we can show that if $a_{t_2} + b_{t_1} \leq r$, then $G[V(R_{t_2}) \cup (V(Q_{t_1}) \setminus \{q_{t_1}\})]$ contains a path of length at most r from y_{t_2} to z_{t_1} . But this is a contradiction and we conclude that $\text{mim}_H(A) \leq w$. \square

Theorem 5.9. *Let r and w be positive integers and G be a graph of clique-width w . Then the r -th power of G has mim-width at most w . Furthermore, given a clique-width w -expression, we can output a branch decomposition of mim-width at most w in polynomial time.*

Proof. Let H be the r -th power of G and let ϕ be the given clique-width w -expression defining G , and T be the syntactic tree of ϕ with root node \mathbf{r} . We can assume that G contains at least two vertices which implies that T has at least one join node. Let \mathbf{r}' be the join node in T with minimum $\text{dist}_T(\mathbf{r}, \mathbf{r}')$. Note that if \mathbf{r} itself is a join node, then $\mathbf{r}' = \mathbf{r}$. Note also that for every vertex v of G , there is a node of T creating v , see the first operation in the definition of clique-width. In the following, we call such a node an *introduce node*. We obtain a branch decomposition (T', \mathcal{L}) as follows:

- We obtain T' from T as follows: If $\mathbf{r} \neq \mathbf{r}'$, we first remove all vertices in the path from \mathbf{r}' to \mathbf{r} in T other than \mathbf{r}' . We fix \mathbf{r}' to be the root node of T' .
- For each introduce node ℓ_v introducing a vertex v , we assign $\mathcal{L}(v) := \ell_v$.

For the first step, note that if the root \mathbf{r} of T is not a join node, then it must have degree one in T . This implies that T' is connected, and that each introduce node is a descendant of \mathbf{r}' . Consider a cut $(V_t, \overline{V_t})$ for some $t \in V(T')$, where V_t is the set of vertices that are introduced below t in T , and note that by construction this is also the set of vertices of G that are mapped to leaves in the subtree of T' rooted at t . If t is a leaf node, then $\text{mim}_H(V_t) \leq 1$. Assume t is an internal node. Then t also appears in T .

We observe that $G[V_t]$ is a w -labeled graph such that for any pair of vertices x, y in the same label class, x and y have the same neighborhood in $V(G) \setminus V_t$. So we can apply Lemma 5.8 to conclude that we have $\text{mim}_H(V_t) \leq w$ which implies that H has mim-width at most w . \square

5.3 Leaf Power Graphs

It is easy to see that trees admit nice tree decompositions all of whose join bags have size 1 and since every leaf power graph is an induced subgraph of a power of some tree, it has mim-width at most 1 by Theorem 5.6. As we argue below, this bound also applies to the following relative of leaf power graphs.

MIN LEAF POWER. Let $r \in \mathbb{N}$. A graph G is an r -min leaf power if there exists a tree T , called a leaf root, whose leaf set is $V(G)$ such that $uv \in E$ if and only if the distance between u and v in T is *more than* r . A graph is called a *min leaf power* if it is an r -min leaf power for some $r \in \mathbb{N}$.

It is not difficult to see that a graph is an r -leaf power if and only if its complement is an r -min leaf power. Together with [281, Lemma 3.7.3], stating that the mim-width of a graph is 1 if and only if the mim-width of its complement is 1, we have the following result.

Corollary 5.10. $\text{mimw}(\text{LEAF POWER}) = \text{mimw}(\text{MIN LEAF POWER}) = 1$.

This result sheds new light on some interesting open questions in graph theory. Leaf powers are known to be strongly chordal graphs and there has recently been interest in delineating the difference between strongly chordal graphs and leaf power graphs, on the assumption that this difference was not very big [198, 232]. Concretely, Lafond stated that “[l]eaf powers are known to be strongly chordal, but few strongly chordal graphs are known to *not* be leaf powers, as such graphs are difficult to construct.” [198, abstract]. Our result actually implies a large difference, as it was recently shown by Mengel that there are strongly chordal split graphs of mim-width linear in the number of vertices [220].

It is a long-standing open problem to decide whether leaf power graphs can be recognized in polynomial time [50, 64, 198, 232]. So far, polynomial-time algorithms for r -leaf power recognition are only known for $r \leq 5$ [71, 234]. Recently, Eppstein and Havvaei showed that the problem of recognizing d -degenerate r -leaf powers is FPT parameterized by $d + r$ [113].

There have been attempts in exploiting the (assumed) closeness between leaf powers and strongly chordal graphs to obtain polynomial-time algorithms to recognize leaf power graphs. Concretely, Nevries and Rosenke [232] asked if a leaf power recognition algorithm could be obtained via strong chordality (which is recognizable in polynomial time [117]) and an additional finite set of forbidden induced subgraphs. Lafond [198] ruled out such an approach by exhibiting an infinite family of minimal strongly chordal graphs that are not leaf powers.

Perhaps the right direction towards a polynomial-time recognition algorithm for leaf power graphs is via Corollary 5.10. Possibly, with additional structural theorems, one can utilize a recognition algorithm for strongly chordal graphs of mim-width 1 to develop a polynomial-time recognition algorithm for leaf powers. We therefore connected this this open problem to the open problem regarding the recognition of graphs of bounded mim-width, in particular mim-width 1, which has been repeatedly stated (e.g. [19, 168, 281]).

Part II

Computing Width Measures

6

Typical Sequences Revisited

In this part we revisit an old key technique from what currently are the theoretically fastest parameterized algorithms for treewidth and pathwidth, namely the use of *typical sequences*, and give additional structural insights for this technique. In particular, we show a structural lemma, which we call the *Merge Dominator Lemma*. The technique of typical sequences brings with it a partial ordering on sequences of integers, and a notion of possible merges of two integer sequences; surprisingly, the Merge Dominator Lemma states that for any pair of integer sequences there exists a *single* merge that dominates all merges of these integer sequences, and this dominating merge can be found in linear time. On its own, this lemma does not lead to asymptotically faster parameterized algorithms for treewidth and pathwidth, but, as we discuss below, it is a concrete step towards such algorithms.

The notion of typical sequences was introduced independently in 1991 by Lagergren and Arnborg [200] and Bodlaender and Kloks [37]. In both papers, it is a key element in an explicit dynamic programming algorithm that given a tree decomposition of bounded width ℓ , decides if the pathwidth or treewidth of the input graph G is at most a constant k . Lagergren and Arnborg build upon this result and show that the set of forbidden minors of graphs of treewidth (or pathwidth) at most k is computable; Bodlaender and Kloks show that the algorithm can also construct a tree or path decomposition of width at most k , if existing, in the same asymptotic time bounds. The latter result is a main subroutine in Bodlaender's linear time algorithm [29] for treewidth- k . Analyzing the runtime of Bodlaender's algorithm for treewidth or pathwidth $\leq k$, one can observe that the bottleneck is in the subroutine that calls the Bodlaender-Kloks dynamic programming subroutine, with both the subroutine and the main algorithm having time $\mathcal{O}(2^{\mathcal{O}(k^3)}n)$ for treewidth, and $\mathcal{O}(2^{\mathcal{O}(k^2)}n)$ for pathwidth. See also the recent work by Fürer for pathwidth [133], and the simplified versions of the algorithms of [29, 37] by Althaus and Ziegler [3]. Now, over a quarter of a century after the discovery of these results, even though much work has been done on treewidth recognition algorithms (see e.g. [4, 34, 44, 118, 133, 199, 251, 256]),

these bounds on the function of k are still the best known, i.e. no $\mathcal{O}(2^{o(k^3)}n^{O(1)})$ algorithm for treewidth, and no $\mathcal{O}(2^{o(k^2)}n^{O(1)})$ algorithm for pathwidth is known. An interesting question, and a long-standing open problem in the field [32, Problem 2.7.1], is whether such algorithms can be obtained. Possible approaches to answer such a question is to design (e.g. ETH or SETH based) lower bounds, find an entirely new approach to compute treewidth or pathwidth in a parameterized setting, or improve upon the dynamic programming algorithms of [200] and [37]. Using our Merge Dominator Lemma we can go one step towards the latter, as follows.

The algorithms of Lagergren and Arnborg [200] and Bodlaender and Kloks [37] are based upon tabulating characteristics of tree or path decompositions of subgraphs of the input graph; a characteristic consists of an *intersection model*, that tells how the vertices in the current top bag interact, and for each *part* of the intersection model, a typical sequence of bag sizes.¹ The set of characteristics for a join node is computed from the sets of characteristics of its (two) children. In particular, each pair of characteristics with one from each child can give rise to exponentially (in k) many characteristics for the join node. This is because *exponentially many* typical sequences may arise as the merges of the typical sequences that are part of the characteristics. In the light of our Merge Dominator Lemma, only *one* of these merges has to be stored, reducing the number of characteristics arising from each pair of characteristics of the children from $2^{\mathcal{O}(k)}$ to just 1. Moreover, this dominating merge can be found in $\mathcal{O}(k)$ time, with no large constants hidden in the ‘ \mathcal{O} ’.

Merging typical sequences at a join node is however not the only way the number of characteristics can increase throughout the algorithm, e.g. at introduce nodes, the number of characteristics increases in a different way. Nevertheless, the number of intersection models is $\mathcal{O}(k^{\mathcal{O}(k)})$ for pathwidth and $\mathcal{O}(k^{\mathcal{O}(k^2)})$ for treewidth; perhaps, with additional techniques, the number of typical sequences per part can be better bounded — in the case that a single dominating typical sequence per part suffices, this would reduce the number of table entries per node to $\mathcal{O}(k^{\mathcal{O}(k)})$ for pathwidth- k , and to $\mathcal{O}(k^{\mathcal{O}(k^2)})$ for treewidth- k , and yield $\mathcal{O}(k^{\mathcal{O}(k)}n)$ and $\mathcal{O}(k^{\mathcal{O}(k^2)}n)$ time algorithms for the respective problems.

6.1 Typical Sequences

In this section we present some additional notation, formally introduce the notion of typical sequences, and show how to compute a typical sequence in linear time. Moreover, we state some lemmas due to Bodlaender and Kloks [37] that will be helpful in later proofs .

¹This approach was later used in several follow up results to obtain explicit constructive parameterized algorithms for other graph width measures, like cutwidth [273, 274], branchwidth [39], different types of search numbers like linear width [40], and directed vertex separation number [36].

Sequences and Matrices. We denote the elements of a sequence s by $s(1), \dots, s(n)$. We denote the *length* of s by $l(s)$, i.e. $l(s) := n$. For two sequences $a = a(1), \dots, a(m)$ and $b = b(1), \dots, b(n)$, we denote their *concatenation* by

$$a \circ b = a(1), \dots, a(m), b(1), \dots, b(n).$$

For two sets of sequences A and B , we let $A \odot B := \{a \circ b \mid a \in A \wedge b \in B\}$. For a sequence s of length n and a set $X \subseteq [n]$, we denote by $s[X]$ the *subsequence of s induced by X* , i.e. let $X = \{x_1, \dots, x_m\}$ be such that for all $i \in [m-1]$, $x_i < x_{i+1}$; then, $s[X] := s(x_1), \dots, s(x_m)$. For $x_1, x_2 \in [n]$ with $x_1 \leq x_2$, we use the shorthand ‘ $s[x_1..x_2]$ ’ for ‘ $s[[x_1..x_2]]$ ’.

Let Ω be a set. A *matrix* $M \in \Omega^{m \times n}$ over Ω is said to have m rows and n columns. For sets $X \subseteq [m]$ and $Y \subseteq [n]$, we denote by $M[X, Y]$ the *submatrix of M induced by X and Y* , which consists of all the entries from M whose indices are in $X \times Y$. For $x_1, x_2 \in [m]$ with $x_1 \leq x_2$ and $y_1, y_2 \in [n]$ with $y_1 \leq y_2$, we use the shorthand ‘ $M[x_1..x_2, y_1..y_2]$ ’ for ‘ $M[[x_1..x_2], [y_1..y_2]]$ ’. For a sequence $s(1), s(2), \dots, s(\ell)$ of indices of a matrix M , we let

$$M[s] := M[s(1)], M[s(2)], \dots, M[s(\ell)] \tag{6.1}$$

be the corresponding sequence of entries from M .

For illustrative purposes we enumerate the columns of a matrix in a bottom-up fashion throughout this part, i.e. we consider the index $(1, 1)$ as the ‘bottom left corner’ and the index (m, n) as the ‘top right corner’.

Integer Sequences. Let s be an integer sequence of length n . We use the shorthand ‘ $\min(s)$ ’ for ‘ $\min_{i \in [n]} s(i)$ ’ and ‘ $\max(s)$ ’ for ‘ $\max_{i \in [n]} s(i)$ ’; we use the following definitions. We let

$$\operatorname{argmin}(s) := \{i \in [n] \mid s(i) = \min(s)\} \text{ and } \operatorname{argmax}(s) := \{i \in [n] \mid s(i) = \max(s)\}$$

be the set of indices at whose positions there are the minimum and maximum element of s , respectively. Whenever we write $i \in \operatorname{argmin}(s)$ ($j \in \operatorname{argmax}(s)$), then the choice of i (j) can be arbitrary. In some places we require a canonical choice of the position of a minimum or maximum element, in which case we will always choose the smallest index. Formally, we let

$$\operatorname{argmin}^*(s) := \min \operatorname{argmin}(s), \text{ and } \operatorname{argmax}^*(s) := \min \operatorname{argmax}(s).$$

The following definition contains two notions on pairs of integer sequences that are necessary for the definitions of domination and merges.

Definition 6.1. Let r and s be two integer sequences of the same length n .

- (i) If for all $i \in [n]$, $r(i) \leq s(i)$, then we write ‘ $r \leq s$ ’.

- (ii) We write $q = r + s$ for the integer sequence $q(1), \dots, q(n)$ with $q(i) = r(i) + s(i)$ for all $i \in [n]$.

Definition 6.2 (Extensions). Let s be a sequence of length n . We define the set $E(s)$ of *extensions* of s as the set of sequences that are obtained from s by repeating each of its elements an arbitrary number of times, and at least once. Formally, we let

$$E(s) := \{s_1 \circ s_2 \circ \dots \circ s_n \mid \forall i \in [n]: l(s_i) \geq 1 \wedge \forall j \in [l(s_i)]: s_i(j) = s(i)\}.$$

Definition 6.3 (Domination). Let r and s be integer sequences. We say that r *dominates* s , in symbols ' $r \prec s$ ', if there are extensions $r^* \in E(r)$ and $s^* \in E(s)$ of the same length such that $r^* \leq s^*$. If $r \prec s$ and $s \prec r$, then we say that r and s are *equivalent*, and we write $r \equiv s$.

If r is an integer sequence and S is a set of integer sequences, then we say that r *dominates* S , in symbols ' $r \prec S$ ', if for all $s \in S$, $r \prec s$.

Remark 6.4 (Transitivity of ' \prec '). In [37, Lemma 3.7], it is shown that the relation ' \prec ' is transitive. As this is fairly intuitive, we may use this fact without stating it explicitly throughout this text.

A *merge* of two integer sequence r and s is the sum of an extension of r with an extension of s of the same length.

Definition 6.5 (Merges). Let r and s be two integer sequences. We define the set of all *merges* of r and s , denoted by $r \oplus s$, as $r \oplus s := \{r^* + s^* \mid r^* \in E(r), s^* \in E(s), l(r^*) = l(s^*)\}$.

Typical Sequences. We now define typical sequences, show how to construct them in linear time, and restate several lemmas due to Bodlaender and Kloks [37] that will be used throughout this text.

Definition 6.6. Let $s = s(1), \dots, s(n)$ be an integer sequence of length n . The *typical sequence* of s , denoted by $\tau(s)$, is obtained from s by an exhaustive application of the following two operations:

Removal of Consecutive Repetitions. If there is an index $i \in [n - 1]$ such that $s(i) = s(i + 1)$, then we change the sequence s from $s(1), \dots, s(i), s(i + 1), \dots, s(n)$ to $s(1), \dots, s(i), s(i + 2), \dots, s(n)$.

Typical Operation. If there exist $i, j \in [n]$ such that $j - i \geq 2$ and for all $i \leq k \leq j$, $s(i) \leq s(k) \leq s(j)$, or for all $i \leq k \leq j$, $s(i) \geq s(k) \geq s(j)$, then we change the sequence s from $s(1), \dots, s(i), s(i + 1), \dots, s(j), \dots, s(n)$ to $s(1), \dots, s(i), s(j), \dots, s(n)$, i.e. we remove all elements (strictly) between index i and j .

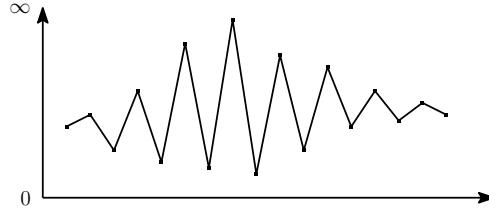


Figure 6.1: Illustration of the shape of a typical sequence.

To support intuition, we illustrate the rough shape of a typical sequence in Figure 6.1. It is not difficult to see that the typical sequence can be computed in quadratic time, by an exhaustive application of the definition. Here we discuss how to do it in linear time. We may view a typical sequence $\tau(s)$ of an integer sequence s as a subsequence of s . While $\tau(s)$ is unique, the choice of indices that induce $\tau(s)$ may not be unique. We show that we can find a set of indices that induce the typical sequence in linear time, with help of the following structural proposition.

Proposition 6.7. *Let s be an integer sequence and let $i^* \in \{\text{argmin}^*(s), \text{argmax}^*(s)\}$. Let $1 =: j_0 < j_1 < j_2 < \dots < j_t < j_{t+1} := i^*$ be pairwise distinct integers, such that $s(j_0), \dots, s(j_{t+1})$ are pairwise distinct. If for all $h \in [0..t]$,*

- if $s(j_h) > s(j_{h+1})$ then $j_h = \text{argmax}^*(s[1..j_{h+1}])$ and $j_{h+1} = \text{argmin}^*(s[1..j_{h+1}])$, and
 - if $s(j_h) < s(j_{h+1})$ then $j_h = \text{argmin}^*(s[1..j_{h+1}])$ and $j_{h+1} = \text{argmax}^*(s[1..j_{h+1}])$,
- then the typical sequence of s restricted to $[i^*]$ is equal to $s(j_0), s(j_1), \dots, s(j_t), s(j_{t+1})$.

Proof. First, we observe that by the choice made in the definition of argmin^* and argmax^* ,

$$\text{for each } h \in [0..(t+1)] \text{ there is no } i < j_h \text{ such that } s(i) = s(j_h). \quad (6.2)$$

We prove the following statement. Under the stated conditions, for a given $h \in [0..t+1]$, the typical sequence of s restricted to $[j_h..i^*]$ is equal to $s(j_h), s(j_{h+1}), \dots, s(j_{t+1})$. The proposition then follows from the case $h = 0$. The proof is by induction on $d := (t+1) - h$. For $d = 0$, it trivially holds since the minimum and the maximum element are always part of the typical sequence, and since $[j_{t+1}..i^*] = \{i^*\}$.

Now suppose $d > 0$, and for the induction hypothesis, that the claim holds for $d - 1$. Suppose that $s(j_h) > s(j_{h+1})$, meaning that $j_h = \text{argmax}^*(s[1..j_{h+1}])$, and $j_{h+1} = \text{argmin}^*(s[1..j_{h+1}])$, the other case is symmetric. By the induction hypothesis, the typical sequence of s restricted to $[j_{h+1}..i^*]$ is equal to $s(j_{h+1}), \dots, s(j_{t+1})$, in particular it implies that $s(j_{h+1})$ is an element of the typical sequence. To prove the induction step, we have to show that the typical sequence of s restricted to $[j_h..j_{h+1}]$

is equal to $s(j_h)$, $s(j_{h+1})$. We first argue that if there is an element of the typical sequence of s in $[j_h..(j_{h+1}-1)]$, then it must be equal to $s(j_h)$. By 6.2, we have that there is no $i < j_{h+1}$ such that $s(i) = s(j_{h+1})$, and together with the fact that $s(j_{h+1})$ is the minimum value of $s[1..j_{h+1}]$, $[j_h..(j_{h+1}-1)]$ cannot contain any element of the typical sequence that is equal to $s(j_{h+1})$. Next, since the typical operation removes all elements $i \in [(j_h+1)..(j_{h+1}-1)]$ with $s(j_h) > s(i) > s(j_{h+1})$, and since $j_h = \text{argmax}^*(s[1..j_{h+1}])$, the only elements from $[j_h..(j_{h+1}-1)]$ that the typical sequence may contain have value $s(j_h)$.

It remains to argue that $s(j_h)$ is indeed an element of the typical sequence. Suppose not, then there are indices i, i' with $i < j_h < i'$, such that either $s(i) \leq s(j_h) \leq s(i')$, or $s(i) \geq s(j_h) \geq s(i')$, and we may assume that at least one of the inequalities is strict in each case. For the latter case, since $j_h = \text{argmax}^*(s[1..j_{h+1}])$, we would have that $s(i) = s(j_h)$, which is a contradiction to (6.2). Hence, we may assume that $s(i) \leq s(j_h) \leq s(i')$. There are two cases to consider: $i' \in [(j_h+1)..j_{h+1}]$, and $i' > j_{h+1}$. If $i' \in [(j_h+1)..j_{h+1}]$, then $s(i') = s(j_h)$, as $s(j_h) = \text{argmax}(s[1..j_{h+1}])$. We can conclude that in this case, the typical sequence must contain an element equal to $s(i')$, and hence equal to $s(j_h)$. If $i' > j_{h+1}$, then the typical operation corresponding to i and i' also removes $s(j_{h+1})$, a contradiction with the induction hypothesis which asserts that $s(j_{h+1})$ is part of the typical sequence induced by $[j_{h+1}..i^*]$. We can conclude that $s(j_h)$ is part of the typical sequence, finishing the proof. \square

From the previous proposition, we have the following consequence about the structure of typical sequences ending in the minimum element, which will be useful in the proof of Lemma 6.20.

Corollary 6.8. *Let t be a typical sequence of length n such that $n \in \text{argmin}(t)$. Then, for each $k \in [\lfloor \frac{n}{2} \rfloor], n-2k+1 \in \text{argmax}(t[1..(n-2k+1)])$ and $n-2k \in \text{argmin}(t[1..(n-2k)])$.*

Equipped with Proposition 6.7, we can now proceed and give the linear-time algorithm that computes a typical sequence of an integer sequence.

Lemma 6.9. *Let s be an integer sequence of length n . Then, one can compute $\tau(s)$, the typical sequence of s , in time $\mathcal{O}(n)$.*

Proof. First, we check for each $i \in [n-1]$ whether $s(i) = s(i+1)$, and if we find such an index i , we remove $s(i)$. We assume from now on that after these modifications, s has at least two elements, otherwise it is trivial. As observed above, the typical sequence of s contains $\min(s)$ and $\max(s)$. A closer look reveals the following observation.

Observation 6.9.1. *Let $i^* := \min \text{argmin}(s) \cup \text{argmax}(s)$ and $k^* := \max \text{argmin}(s) \cup \text{argmax}(s)$.*

- (i) *If $i^* \in \text{argmin}(s)$ and $k^* \in \text{argmax}(s)$ or $i^* \in \text{argmax}(s)$ and $k^* \in \text{argmin}(s)$, then $\tau(s)$ restricted to $[i^*..k^*]$ is $s(i^*), s(k^*)$.*

- (ii) If $\{i^*, k^*\} \subseteq \text{argmin}(s)$, then $\tau(s)$ restricted to $[i^*..k^*]$ is $s(i^*), \max(s), s(k^*)$.
 (iii) If $\{i^*, k^*\} \subseteq \text{argmax}(s)$, then $\tau(s)$ restricted to $[i^*..k^*]$ is $s(i^*), \min(s), s(k^*)$.

Let $i^* := \min \text{argmin}(s) \cup \text{argmax}(s)$ and $k^* := \max \text{argmin}(s) \cup \text{argmax}(s)$. Using Observation 6.9.1, it remains to determine the indices that induce the typical sequence on $s[1..i^*]$ and on $s[k^*..n]$. To find the indices that induce the typical sequence on $s[1..i^*]$, we will describe a marking procedure that marks a set of indices satisfying the preconditions of Proposition 6.7. Next, we observe that $n - k^*$ is the *smallest* index of any occurrence of $\min(s)$ or $\max(s)$ in the *reverse* sequence of s , therefore a symmetric procedure, again using Proposition 6.7, yields the indices that induce $\tau(s)$ on $s[k^*..n]$.

```

1  $j_{\min} \leftarrow \text{argmin}^*(s[1..2]), j_{\max} \leftarrow \text{argmax}^*(s[1..2]), M \leftarrow \{1\}$ 
2 for  $j = 3, \dots, i^*$  do
3   if  $s(j) < s(j_{\min})$  then
4      $j_{\min} \leftarrow j$ 
5      $M \leftarrow M \cup \{j_{\max}\}$  // mark the current value of  $j_{\max}$ 
6   if  $s(j) > s(j_{\max})$  then
7      $j_{\max} \leftarrow j$ 
8      $M \leftarrow M \cup \{j_{\min}\}$  // mark the current value of  $j_{\min}$ 
9  $M \leftarrow M \cup \{j_{\min}, j_{\max}\}$ 
```

Algorithm 3: The algorithm of Lemma 6.9 that computes the set M of indices that induce the typical sequence of s between the first element and the first occurrence of the minimum and maximum of s .

We execute Algorithm 3, which processes the integer sequence $s[1..i^*]$ from the first to the last element, storing two counters j_{\min} and j_{\max} that store the leftmost position of the smallest and of the greatest element seen so far, respectively. Whenever a new minimum is encountered, we mark the current value of the index j_{\max} , as this implies that $s(j_{\max})$ has to be an element of the typical sequence. Similarly, when encountering a new maximum, we mark j_{\min} . These marked indices are stored in a set M , which at the end of the algorithm contains the indices that induce $\tau(s)$ on $[1..i^*]$. This, i.e. the correctness of the procedure, will now be argued via Proposition 6.7.

Claim 6.9.2. *The set M of indices marked by the above procedure induce $\tau(s)$ on $[1..i^*]$.*

Proof. Let $M = \{j_0, j_1, \dots, j_{t+1}\}$ be such that for all $h \in [0..t]$, $j_h < j_{h+1}$. We prove that j_0, \dots, j_{t+1} meet the preconditions of Proposition 6.7. First, we observe that the above algorithm marks both the index 1 and index i^* , in particular that $j_0 = 1$ and $j_{t+1} = i^*$.

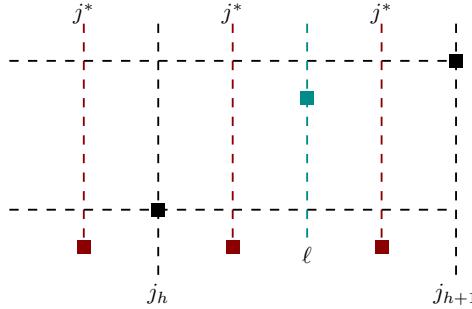


Figure 6.2: Illustration of the final argument in the proof of Claim 6.9.2. We assume that $s(j_h) < s(j_{h+1})$, and mark the possible positions for $j^* = \operatorname{argmin}^*(s[1..j_{h+1}])$ with $j^* \neq j_h$.

We verify that the indices j_0, \dots, j_{t+1} satisfy the property that for each $h \in [0..(t+1)]$, the index j_h is the leftmost (i.e. smallest) index whose value is equal to $s(j_h)$: whenever an index is added to the marked set, it is because in some iteration, the element at its position was either strictly greater than the greatest previously seen element, or strictly smaller than the smallest previously seen element. (This also ensures that $s(j_0), \dots, s(j_{t+1})$ are pairwise distinct.)

We additionally observe that if we have two indices ℓ_1 and ℓ_2 such that ℓ_2 is the index that the algorithm marked right after it marked ℓ_1 , then either ℓ_1 was j_{\min} and ℓ_2 was j_{\max} or vice versa: when updating j_{\min} , we mark j_{\max} , and when updating j_{\max} , we mark j_{\min} . This lets us conclude that when we have two indices j_h, j_{h+1} such that $s(j_h) < s(j_{h+1})$, then j_h was equal to j_{\min} when it was marked, and j_{h+1} was j_{\max} when it was marked.

We are ready to prove that j_0, \dots, j_{t+1} satisfy the precondition of Proposition 6.7. Suppose for a contradiction that for some $h \in [0..t+1]$, j_h violates this property. Assume that $s(j_h) < s(j_{h+1})$ and note that the other case is symmetric. The previous paragraph lets us conclude that j_h was equal to j_{\min} when it was marked, and that j_{h+1} was j_{\max} when it was marked.

Either $j_h \neq \operatorname{argmin}^*(s[1..j_{h+1}])$ or $j_{h+1} \neq \operatorname{argmax}^*(s[1..j_{h+1}])$. Suppose the latter holds. This immediately implies that there is some $j^* \in [j_{h+1} - 1]$ such that $s(j^*) > j_{h+1}$, which implies that j_{\max} would never have been set to j_{h+1} and hence j_{h+1} would have never been marked. Suppose the former holds, i.e. $j_h \neq \operatorname{argmin}^*(s[1..j_{h+1}])$, for an illustration of the following argument see Figure 6.2. Let $j^* := \operatorname{argmin}^*(s[1..j_{h+1}])$. If $j^* < j_h$, then at iteration j_h , $s(j_{\min}) < s(j_h)$, so j_{\min} would never have been set to j_h , and hence, j_h would never have been marked. We may assume that $j^* > j_h$. Since j_h was marked, there is some $\ell > j_h$ that triggered j_h being marked. This also means that at that iteration $s(\ell)$ was greater than the previously observed maximum, so we may assume that $s(\ell) > s(j_h)$. We also may assume that $\ell \leq j_{h+1}$. If $j^* \in [(j_h + 1)..(\ell - 1)]$, then the algorithm

would have updated j_{\min} to j^* in that iteration, before marking j_h , and for the case $j^* \in [(\ell + 1)..(j_{h+1} - 1)]$ we observe that $\ell \neq j_{h+1}$, and that the algorithm would mark ℓ as the next index instead of j_{h+1} . \square

This establishes the correctness of the algorithm. For its runtime, we observe that each iteration takes $\mathcal{O}(1)$ time, and that there are $\mathcal{O}(n)$ iterations. \square

We summarize several lemmas from [37] regarding integer sequences and typical sequences that we will use in this work.

Lemma 6.10 (Bodlaender and Kloks [37]). *Let r and s be two integer sequences.*

- (i) (Cor. 3.11 in [37]). *We have that $r \prec s$ if and only if $\tau(r) \prec \tau(s)$.*
- (ii) (Lem. 3.13 in [37]). *Suppose r and s are of the same length and let $y = r + s$. Let $r_0 \prec r$ and $s_0 \prec s$. Then there is an integer sequence $y_0 \in r_0 \oplus s_0$ such that $y_0 \prec y$.*
- (iii) (Lem. 3.14 in [37]). *Let $q \in r \oplus s$. Then, there is an integer sequence $q' \in \tau(r) \oplus \tau(s)$ such that $q' \prec q$.*
- (iv) (Lem. 3.15 in [37]). *Let $q \in r \oplus s$. Then, there is an integer sequence $q' \in r \oplus s$ with $\tau(q') = \tau(q)$ and $l(q') \leq l(r) + l(s) - 1$.*
- (v) (Lem. 3.19 in [37]). *Let r' and s' be two more integer sequences. If $r' \prec r$ and $s' \prec s$, then $r' \circ s' \prec r \circ s$.*

6.2 The Merge Dominator Lemma

In this section we prove the *Merge Dominator Lemma* which states that given two integer sequences, one can find in linear time a merge that dominates all merges of those two sequences.

Lemma 6.11 (Merge Dominator Lemma). *Let r and c be integer sequence of length m and n , respectively. There exists a dominating merge of r and c , i.e. an integer sequence $t \in r \oplus c$ such that $t \prec r \oplus c$, and this dominating merge can be computed in time $\mathcal{O}(m + n)$.*

Outline of the proof of the Merge Dominator Lemma. First, we show that we can restrict our search to finding a dominating path in a matrix that, roughly speaking, contains all merges of r and c of length at most $l(r) + l(c) - 1$. The goal of this step is mainly to increase the intuitive insight to the proofs in this chapter. Next, we prove the ‘Split Lemma’ (Lemma 6.17 in Subsection 6.2.2) which asserts that we can obtain a dominating path in our matrix M by splitting M into a submatrix M_1 that lies in the ‘bottom left’ of M and another submatrix M_2 in the ‘top right’

of M along a minimum row and a minimum column, and appending a dominating path in M_2 to a dominating path in M_1 . In M_1 , the last row and column are a minimum row and column, respectively, and in M_2 , the first row and column are a minimum row and column, respectively. This additional structure will be exploited in Subsection 6.2.3 where we prove the ‘Chop Lemmas’ that come in two versions. The ‘bottom version’ (Lemma 6.20) shows that in M_1 , we can find a dominating path by repeatedly chopping away the *last* two rows or columns and remembering a vertical or horizontal length-2 path. The ‘top version’ (Corollary 6.22) is the symmetric counterpart for M_2 . The proofs of the Chop Lemmas only hold when r and c are *typical sequences*, and in Subsection 6.2.4 we present the ‘Split-and-Chop Algorithm’ that computes a dominating path in a merge matrix of two typical sequences. Finally, in Subsection 6.2.5, we generalize this result to arbitrary integer sequences, using the Split-and-Chop Algorithm and one additional construction.

6.2.1 The Merge Matrix, Paths, and Non-Diagonality

Let us begin by defining the basic notions of a merge matrix and paths in matrices.

Definition 6.12 (Merge Matrix). Let r and c be two integer sequences of length m and n , respectively. Then, the *merge matrix* of r and c is an $m \times n$ integer matrix M such that for $(i, j) \in [m] \times [n]$, $M[i, j] = r(i) + c(j)$.

Definition 6.13 (Path in a Matrix). Let M be an $m \times n$ matrix. A *path* in M is a sequence $p(1), \dots, p(\ell)$ of indices from M such that

- (i) $p(1) = (1, 1)$ and $p(\ell) = (m, n)$, and
- (ii) for $h \in [\ell - 1]$, let $p(h) = (i, j)$; then, $p(h+1) \in \{(i+1, j), (i, j+1), (i+1, j+1)\}$.

We denote by $\mathcal{P}(M)$ the set of all paths in M . A sequence $p(1), \dots, p(\ell)$ that satisfies the second condition but not necessarily the first is called a *partial path* in M . For two paths $p, q \in \mathcal{P}(M)$, we may simply say that p *dominates* q , if $M[p]$ dominates $M[q]$.² We also write $p \prec \mathcal{P}(M)$ to express that for each path $q \in \mathcal{P}(M)$, $p \prec q$.

A (partial) path is called *non-diagonal* if the second condition is replaced by the following.

- (ii)' For $h \in [\ell - 1]$, let $p(h) = (i, j)$; then, $p(h + 1) \in \{(i + 1, j), (i, j + 1)\}$.

An *extension* e of a path p in a matrix M is as well a sequence of indices of M , and we again denote the corresponding integer sequence by $M[e]$. A consequence of Lemma 6.10(i) and (iv) is that we can restrict ourselves to all paths in a merge matrix when trying to find a dominating merge of two integer sequences: it is clear from the definitions that in a merge matrix M of integer sequences r and c , $\mathcal{P}(M)$

²Recall that by (6.1) on page 101, for a (partial) path p in a matrix M , $M[p] = M[p(1)], M[p(2)], \dots, M[p(l(p))]$.

contains all merges of r and c of length at most $l(r) + l(c) - 1$. Furthermore, suppose that there is a merge $q \in r \oplus s$ such that $q \prec r \oplus s$ and $l(q) > l(r) + l(s) - 1$. By Lemma 6.10(iv), there is a merge $q' \in r \oplus s$ such that $l(q') \leq l(r) + l(s) - 1$, and $\tau(q') = \tau(q)$. The latter yields $\tau(q') \equiv \tau(q)$ and therefore, by Lemma 6.10(i), $q' \equiv q$, in particular, $q' \prec q \prec r \oplus s$.

Corollary 6.14. *Let r and c be integer sequences and M be the merge matrix of r and c . There is a dominating merge in $r \oplus c$, i.e. an integer sequence $t \in r \oplus c$ such that $t \prec r \oplus c$, if and only if there is a dominating path in M , i.e. a path $p \in \mathcal{P}(M)$ such that $p \prec \mathcal{P}(M)$.*

We now consider a type of merge that corresponds to non-diagonal paths in the merge matrix. These merges will be used in a construction presented in Subsection 6.2.5, and in the algorithmic applications of the Merge Dominator Lemma given in chapter 7. For two integer sequences r and s , we denote by $r \boxplus s$ the set of all *non-diagonal merges* of r and s , which are not allowed to have ‘diagonal’ steps: we have that for all $t \in r \boxplus s$ and all $i \in [l(t) - 1]$, if $t(i) = r(i_r) + s(i_s)$, then $t(i+1) \in \{r(i_r+1) + s(i_s), r(i_r) + s(i_s+1)\}$. As each non-diagonal merge directly corresponds to a non-diagonal path in the merge matrix (and vice versa), we can consider a non-diagonal path in a merge matrix to be a non-diagonal merge and vice versa. We now show that for each merge that uses diagonal steps, there is always a non-diagonal merge that dominates it.

Lemma 6.15. *Let r and s be two integer sequences of length m and n , respectively. For any merge $q \in r \oplus s$, there is a non-diagonal merge $q' \in r \boxplus s$ such that $q' \prec q$. Furthermore, given q , q' can be found in time $\mathcal{O}(m+n)$.*

Proof. This can be shown by the following local observation. Let $i \in [l(q) - 1]$ be such that $q(i), q(i+1)$ is a diagonal step, i.e. there are indices $i_r \in [l(r) - 1]$ and $i_s \in [l(s) - 1]$ such that $q(i) = r(i_r) + s(i_s)$ and $q(i+1) = r(i_r+1) + s(i_s+1)$. Then, we insert the element $x := \min\{r(i_r) + s(i_s+1), r(i_r+1) + s(i_s)\}$ between $q(i)$ and $q(i+1)$. Since

$$x \leq \max\{r(i_r) + s(i_s), r(i_r+1), s(i_s+1)\} =: y,$$

we can repeat y twice in an extension of q so that one of the occurrences aligns with x , and we have that in this position, the value of q' is at most the value of the extension of q .

Let q' be the sequence obtained from q by applying this operation to all diagonal steps, then by the observation just made, we have that $q' \prec q$. It is clear that this can be implemented to run in time $\mathcal{O}(m+n)$. \square

Next, we define two special paths in a matrix M that will reappear in several places throughout this chapter. These paths can be viewed as the ‘corner paths’, where the first one follows the first row until it hits the last column and then follows

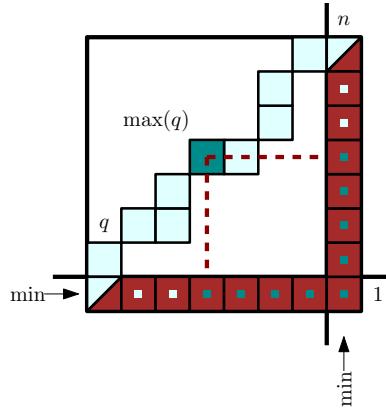


Figure 6.3: Situation in the proof of Lemma 6.16(i). The dot within each element of the corner path p_{\llcorner} indicates with which elements of the path q it is ‘matched up’ in the extensions constructed in the proof.

the last column ($p_{\llcorner}(M)$), and the second one follows the first column until it hits the last row and then follows the last row ($p^-(M)$). Formally, we define them as follows:

$$\begin{aligned} p_{\llcorner}(M) &:= (1, 1), (1, 2), \dots, (1, n), (2, n), \dots, (m, n) \\ p^-(M) &:= (1, 1), (2, 1), \dots, (m, 1), (m, 2), \dots, (m, n) \end{aligned}$$

We use the shorthands ‘ p_{\llcorner} ’ for ‘ $p_{\llcorner}(M)$ ’ and ‘ p^- ’ for ‘ $p^-(M)$ ’ whenever M is clear from the context.

For instance, these paths appear in the following special cases of the Merge Dominator Lemma, which will be useful for several proofs in this section.

Lemma 6.16. *Let r and c be integer sequences of length m and n , respectively, and let M be the merge matrix of r and c . Let $i \in \text{argmin}(r)$ and $j \in \text{argmin}(c)$.*

- (i) *If $i = 1$ and $j = n$, then p_{\llcorner} dominates all paths in M , i.e. $p_{\llcorner} \prec \mathcal{P}(M)$.*
- (ii) *If $i = m$ and $j = 1$, then p^- dominates all paths in M , i.e. $p^- \prec \mathcal{P}(M)$.*

Proof. (i) For an illustration of this proof see Figure 6.3. Let q be any path in M and let $t^* := \text{argmax}^*(q)$. Let furthermore $q(t^*) = (t_r^*, t_c^*)$. We divide p_{\llcorner} and q in three consecutive parts each to show that p_{\llcorner} dominates q .

- We let $p_{\llcorner}^1 := p_{\llcorner}(1), \dots, p_{\llcorner}(t_c^* - 1)$ and $q_1 := q(1), \dots, q(t^* - 1)$.
- We let $p_{\llcorner}^2 := p_{\llcorner}(t_c^*), \dots, p_{\llcorner}(n + t_r^* - 1)$ and $q_2 := q(t^*)$.
- We let $p_{\llcorner}^3 := p_{\llcorner}(n + t_r^*), \dots, p_{\llcorner}(m + n - 1)$ and $q_3 := q(t^* + 1), \dots, q(l(q))$.

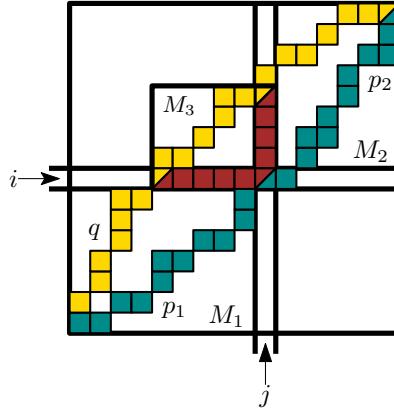


Figure 6.4: Situation in the proof of Lemma 6.17.

Since $r(1)$ is a minimum row in M , we have that for all $(k, \ell) \in [m] \times [n]$, $M[1, \ell] \leq M[k, \ell]$. This implies that there is an extension e_1 of p_1^1 of length $t^* - 1$ such that $M[e_1] \leq M[q_1]$. Similarly, there is an extension e_3 of p_3^3 of length $l(q) - t^*$ such that $M[e_3] \leq M[q_3]$. Finally, let f_2 be an extension of q_2 that repeats its only element, $q(t^*)$, $n - t_c^* + t_r^*$ times. Since $M[q(t^*)]$ is the maximum element on the sequence $M[q]$ and $r(1)$ is a minimum row and $c(n)$ a minimum column in M , we have that $M[p_2^2] \leq M[f_2]$: for all $h \in [t_c^*..n]$, there is some $h_q \in [l(q)]$ such that $q(h_q) = (j, h)$ for some row j , so $M[p_2^2(h)] = M[1, h] \leq M[j, h] = M[q(h_q)] \leq M[q(t^*)]$ (similarly for all $h \in [n..(n + t_r^* - 1)]$).

We define an extension e of p_\perp as $e := e_1 \circ p_\perp^2 \circ e_3$ and an extension f of q as $f := q_1 \circ f_2 \circ q_3$. Note that $l(e) = l(f) = l(q) + n + t_r^* - (t_c^* + 1)$, and by the above discussion, we have that $M[e] \leq M[f]$. (ii) follows from a symmetric argument. \square

6.2.2 The Split Lemma

In this section we prove the first main step towards the Merge Dominator Lemma. It is fairly intuitive that a dominating merge has to contain the minimum element of a merge matrix. (Otherwise, there is a path that cannot be dominated by that merge.) The Split Lemma states that in fact, we can split the matrix M into two smaller submatrices, one that has the minimum element in the top right corner, and one that has the minimum element in the bottom left corner, compute a dominating path for each of them, and paste them together to obtain a dominating path for M .

Lemma 6.17 (Split Lemma). *Let r and c be integer sequences of length m and n , respectively, and let M be the merge matrix of r and c . Let $i \in \text{argmin}(r)$ and $j \in \text{argmin}(c)$. Let $M_1 := M[1..i, 1..j]$ and $M_2 := M[i..m, j..n]$ and for all $h \in [2]$, let $p_h \in \mathcal{P}(M_h)$ be a dominating path in M_h , i.e. $p_h \prec \mathcal{P}(M_h)$. Then, $p_1 \circ p_2$ is a*

dominating path in M , i.e. $p_1 \circ p_2 \prec \mathcal{P}(M)$.

Proof. Let q be any path in M . If q contains (i, j) , then q has two consecutive parts, say q_1 and q_2 , such that $q_1 \in \mathcal{P}(M_1)$ and $q_2 \in \mathcal{P}(M_2)$. Hence, $p_1 \prec q_1$ and $p_2 \prec q_2$, so by Lemma 6.10(v), $p_1 \circ p_2 \prec q_1 \circ q_2$.

Now let $p := p_1 \circ p_2$ and suppose q does not contain (i, j) . Then, q either contains some (i, j') with $j' < j$, or some (i', j) , with $i' < i$. We show how to construct extensions of p and q that witness that p dominates q in the first case, and remark that the second case can be shown symmetrically. We illustrate this situation in Figure 6.4.

Suppose that q contains (i, j') with $j' < j$. We show that $p \prec q$. First, q also contains some (i', j) , where $i' > i$. Let h_1 be the index of (i, j') in q , i.e. $q(h_1) = (i, j')$, and h_2 denote the index of (i', j) in q , i.e. $q(h_2) = (i', j)$. We derive the following sequences from q .

- We let $q_1 := q(1), \dots, q(h_1)$ and $q_1^+ := q_1 \circ (i, j'+1), \dots, (i, j)$.
- We let $q_{12} := q(h_1), \dots, q(h_2)$.
- We let $q_2 := q(h_2), \dots, q(l(q))$ and $q_2^+ := (i, j), (i+1, j), \dots, (i', j) \circ q_2$.

Since $q_1^+ \in \mathcal{P}(M_1)$ and $p_1 \prec \mathcal{P}(M_1)$, we have that $p_1 \prec q_1^+$, similarly that $p_2 \prec q_2^+$ and considering $M_3 := M[i'..i, j..j']$, we have by Lemma 6.16(i) that $p_{12} := p_1 \circ p_2 \circ (M_3) = (i, j'), (i, j'+1), \dots, (i, j), (i+1, j), \dots, (i', j)$ dominates q_{12} . Consequently, we consider the following extensions of these sequences.

- (I) We let $e_1 \in E(p_1)$ and $f_1 \in E(q_1^+)$ such that $l(e_1) = l(f_1)$ and $M[e_1] \leq M[f_1]$.
- (II) We let $e_{12} \in E(p_{12})$, and $f_{12} \in E(q_{12})$ such that $l(e_{12}) = l(f_{12})$ and $M[e_{12}] \leq M[f_{12}]$.
- (III) We let $e_2 \in E(p_2)$, and $f_2 \in E(q_2^+)$ such that $l(e_2) = l(f_2)$ and $M[e_2] \leq M[f_2]$.

We construct extensions $e' \in E(p)$ and $f' \in E(q)$ as follows. Let z be the last index in q of any element that is matched up with (i, j) in the extensions of (II). (Following the proof of Lemma 6.16, this would mean z is the index of $\max(q_{12})$ in q .) We first construct a pair of extensions $e'_j \in E(p_1)$, and $f'_j \in E(q[1..z])$ with $l(e'_j) = l(f'_j)$ and $M[e'_j] \leq M[f'_j]$. With a symmetric procedure, we can obtain extensions of p_2 and of $q[(z+1)..l(q)]$, and use them to obtain extensions of $p = p_1 \circ p_2$ and $q = q[1..z] \circ q[(z+1)..l(q)]$ witnessing that $p \prec q$.

We give the details of the first part of the construction. Let a be the index of the last repetition in f_1 of $q(h_1 - 1)$, i.e. the index that appears just before $q(h_1) = (i, j')$ in f_1 . We let $e'_{j'-1}[1..a] := e_1[1..a]$ and $f'_{j'-1}[1..a] := f_1[1..a]$. By (I), $M[e'_{j'-1}] \leq M[f'_{j'-1}]$.

For $x = j', j'+1, \dots, j$, we inductively construct e'_x and f'_x using e'_{x-1} and f'_{x-1} , for an illustration see Figure 6.5. We maintain as an invariant that $l(e'_{x-1}) = l(f'_{x-1})$

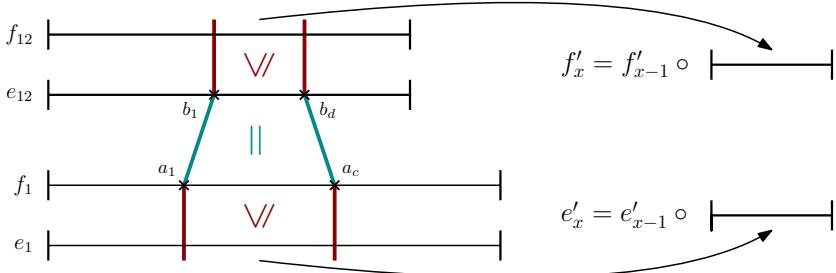


Figure 6.5: Constructing extensions in the proof of Lemma 6.17.

and that $M[e'_{x-1}] \leq M[f'_{x-1}]$. Let a_1, \dots, a_c denote the indices of the occurrences of (i, x) in f_1 , and b_1, \dots, b_d denote the indices of the occurrences of (i, x) in e_{12} . We let:

$$\begin{aligned} e'_x &:= e'_{x-1} \circ e_1[a_1, \dots, a_c] \text{ and } f'_x := f'_{x-1} \circ f_{12}[b_1, \dots, b_d], && \text{if } c = d \\ e'_x &:= e'_{x-1} \circ e_1[a_1, \dots, a_c] \circ \overbrace{e_1(a_c), \dots, e_1(a_c)}^{d-c \text{ times}} \text{ and } f'_x := f'_{x-1} \circ f_{12}[b_1, \dots, b_d], && \text{if } c < d \\ e'_x &:= e'_{x-1} \circ e_1[a_1, \dots, a_c] \text{ and } f'_x := f'_{x-1} \circ f_{12}[b_1, \dots, b_d] \circ \overbrace{f_{12}(b_d), \dots, f_{12}(b_d)}^{c-a \text{ times}}, && \text{if } c > d \end{aligned}$$

In each case, we extended e'_{x-1} and f'_{x-1} by the same number of elements; furthermore we know by (I) that for $y \in \{a_1, \dots, a_c\}$, $M[e_1(y)] \leq M[f_1(y)]$, by choice we have that for all $y' \in \{b_1, \dots, b_d\}$, $f_1(y) = e_{12}(y')$ and we know that $M[e_{12}(y')] \leq M[f_{12}(y')]$ by (II). Hence, $M[e'_x] \leq M[f'_x]$ in either of the above cases. In the end of this process, we have $e'_j \in E(p_1)$ and $f'_j \in E(q[1..z])$, and by construction, $l(e'_j) = l(f'_j)$ and $M[e'_j] \leq M[f'_j]$. \square

6.2.3 The Chop Lemmas

Assume the notation of the Split Lemma. If we were to apply it recursively, it only yields a size reduction whenever $(i, j) \notin \{(1, 1), (m, n)\}$. Motivated by this issue, we prove two more lemmas to deal with the cases when $(i, j) \in \{(1, 1), (m, n)\}$, and we coin them the ‘Chop Lemmas’. It will turn out that when applied to typical sequences, a repeated application of these lemmas yields a dominating path in M . This insight crucially helps in arguing that the dominating path in a merge matrix can be found in *linear* time. Before we present their statements and proofs, we need another auxiliary lemma.

Lemma 6.18. *Let r and c be integer sequences of length m and n , respectively, and let M be the merge matrix of r and c . Let $i \in \text{argmin}(r)$ and $j \in \text{argmin}(c)$. Let furthermore $k \in \text{argmin}(r[\{1, \dots, m\} \setminus \{i\}])$ and $\ell \in \text{argmin}(c[\{1, \dots, n\} \setminus \{j\}])$. Let $\{p^*, q^*\} = \{p_\perp, p^\top\}$ such that $\max(M[p^*]) \leq \max(M[q^*])$.*

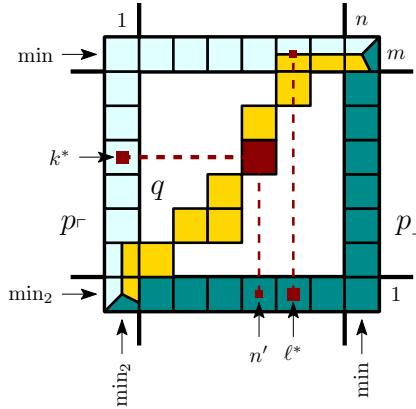


Figure 6.6: Situation in the first stage of the proof of Lemma 6.18(i). The row and column labeled ‘min’ contains the minimum element from the respective sequence, and the row and column labeled ‘min₂’ contains the minimum element among all elements except the one in the min-row or -column.

- (i) If $i = m$, $j = n$, $k = 1$, and $\ell = 1$, then $p^* \prec \mathcal{P}(M)$.
- (ii) If $i = 1$, $j = 1$, $k = m$, and $\ell = n$, then $p^* \prec \mathcal{P}(M)$.

Proof. (i). First, we may assume that $r(1) > r(m)$ and that $c(1) > c(n)$, otherwise we could have applied one of the cases of Lemma 6.16. We prove the lemma in two steps:

1. We show that for each path q in M , p_\perp or p^- (or both) dominate(s) q .
2. We show that p_\perp dominates p^- , or vice versa, or both (depending on which case we are in).

The following claim will be useful in both steps and can be seen as a slight generalization of Lemma 6.16.

Claim 6.18.1. Let $q \in \mathcal{P}(M)$ and let $p \in \{p_\perp, p^-\}$. If $\max(M[p]) \leq \max(M[q])$, then $p \prec q$.

Proof. Suppose that $p = p_\perp$, the other case is symmetric. The claim can be shown using the same argument as in Lemma 6.16, paying slight attention to the situation in which the maximum value of q is in row m , which implies that the maximum of p_\perp is in the same column. \perp

We prove Step 1. For the following argument, see Figure 6.6. If $\max(M[q]) \geq \max(M[p_\perp])$, then we conclude by Claim 6.18.1 that $p_\perp \prec q$ and we are done with

Step 1 of the proof. Suppose

$$\max(M[q]) < \max(M[p_\perp]) \quad (6.3)$$

and let $\ell^* \in \operatorname{argmax}(M[p_\perp])$. We may assume that $\ell^* < n$: otherwise, (6.3) cannot be satisfied since $n \in \operatorname{argmin}(c)$. We furthermore have that q contains (m, ℓ^*) , since p_\perp contains $(1, \ell^*)$, and m is the only position in which r is (potentially) smaller than $r(1)$. Therefore, this is the only way in which (6.3) can be satisfied.

Now let $k^* \in \operatorname{argmax}(M[p_\cap])$. As above, we may assume that $k^* < m$. Now, since q contains (m, ℓ^*) , we have that q also contains (k^*, n') for some $n' < n$. It follows that

$$\max(M[p_\cap]) = M[k^*, 1] \leq M[k^*, n'] \leq \max(M[q])$$

where the first inequality follows from the fact that $r(1) \leq r(n')$ for all $n' < n$. By Claim 6.18.1, p_\cap dominates q and we finished Step 1 of the proof. Step 2 follows from another application of Claim 6.18.1 and the lemma follows from transitivity of the domination relation. This proves (i), and (ii) follows from a symmetric argument. \square

Remark 6.19. We would like to stress that up to this point, all results in this section were shown in terms of arbitrary integer sequences. For the next lemma, we require the sequences considered to be *typical sequences*. In Subsection 6.2.5 we will generalize the results that rely on the following lemmas to arbitrary integer sequences.

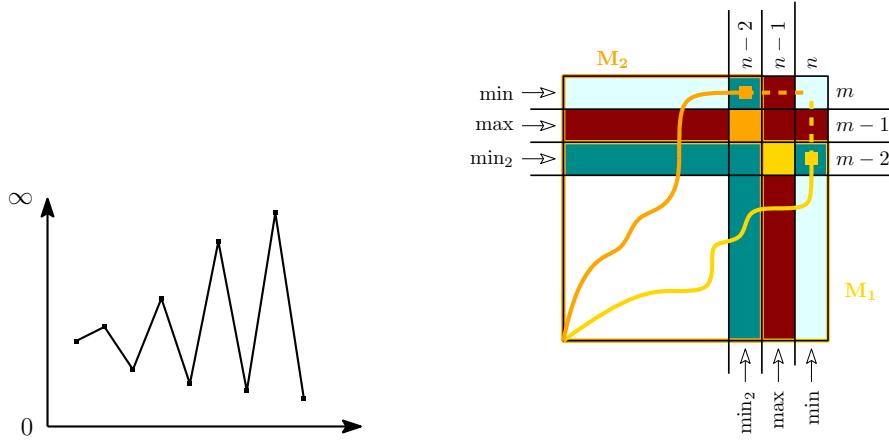
We are now ready to prove the Chop Lemmas for typical sequences. They come in two versions, one that is suited for the case of the bottom left submatrix after an application of the Split Lemma to M , and one for the top right submatrix. In the former case, we have that the last row is a minimum row and that the last column is a minimum column. We will prove this lemma in more detail and observe that the other case follows by symmetry with the arguments given in the following proof. For an illustration of the setting in the following lemma, see Figure 6.7b.

Lemma 6.20 (Chop Lemma - Bottom). *Let r and c be typical sequences of length $m \geq 3$ and $n \geq 3$, respectively, and let M be the merge matrix of r and c . Suppose that $m \in \operatorname{argmin}(r)$ and $n \in \operatorname{argmin}(c)$ and let $M_1 := M[1..(m-2), 1..n]$ and $M_2 := M[1..m, 1..(n-2)]$ and for all $h \in [2]$, let $p_h \prec \mathcal{P}(M_h)$. Let $p_1^+ := p_1 \circ (m-1, n), (m, n)$ and $p_2^+ := p_2 \circ (m, n-1), (m, n)$.*

- (i) *If $M[m-2, n-1] \leq M[m-1, n-2]$, then $p_1^+ \prec \mathcal{P}(M)$.*
- (ii) *If $M[m-1, n-2] \leq M[m-2, n-1]$, then $p_2^+ \prec \mathcal{P}(M)$.*

Proof. Let $s \in \{r, c\}$. Since s is a typical sequence and $l(s) \in \operatorname{argmin}(s)$, we know by Corollary 6.8 that for all $k \in [|l(s)/2|]$,

$$\begin{aligned} l(s) - 2k + 1 &\in \operatorname{argmax}(s[1..(l(s) - 2k + 1)]) \text{ and} \\ l(s) - 2k &\in \operatorname{argmin}(s[1..(l(s) - 2k)]). \end{aligned}$$



(a) Typical sequence ending in the minimum.

(b) The basic setup in Lemma 6.20.

Figure 6.7: Visual aides to the proof of Lemma 6.20.

Informally speaking, this means that the last element of s is the minimum, the $(l(s)-1)$ -th element of s is the maximum, the $(l(s)-2)$ -th element is ‘second-smallest’ element, and so on. We will therefore refer to the element at position $l(s) - 2k$ ($2k \leq l(s)$) as ‘ $\min_{k+1}(s)$ ’ (note that the minimum is achieved when $k = 0$, hence the ‘+1’), and elements at position $l(s) - 2k + 1$ ($2k + 1 \leq l(s) - 1$) as ‘ $\max_k(s)$ ’. For an illustration of the shape of s see Figure 6.7a and for an illustration of the basic setting of this proof see Figure 6.7b. We prove (i) and remark that the argument for (ii) is symmetric.

First, we show that each path in M is dominated by at least one of p_1^+ and p_2^+ .

Claim 6.20.1. Let $q \in \mathcal{P}(M)$. Then, for some $r \in [2]$, $p_r^+ \prec q$.

Proof. We may assume that q does not contain $(m-1, n-1)$: if so, we could easily obtain a path q' from q by some local replacements such that q' dominates q , since $M[m-1, n-1]$ is the maximum element of the matrix M . We may assume that q either contains $(m-1, n)$ or $(m, n-1)$. Assume that the former holds, and note that an argument for the latter case can be given analogously. Since q contains $(m-1, n)$, and since q does not contain $(m-1, n-1)$, we may assume that q contains $(m-2, n)$: if not, we can simply add $(m-2, n)$ before $(m-1, n)$ to obtain a path that dominates q (recall that n is the column indexed by the minimum of c). Now, let $q|_{M_1}$ be the restriction of q to M_1 , we then have that $q = q|_{M_1} \circ (m-1, n), (m, n)$. Since p_1 dominates all paths in M_1 , it dominates $q|_{M_1}$ and so $p_1^+ \prec q$. \square

The remainder of the proof is devoted to showing that p_1^+ dominates p_2^+ which yields the lemma by Claim 6.20.1 and transitivity. To achieve that, we will show in

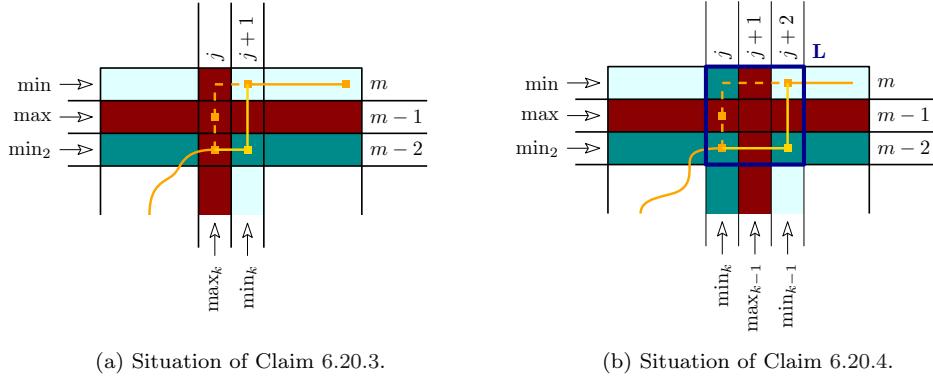


Figure 6.8: Visualization of the arguments that lead to the conclusion that we may assume that p_2 contains $(m-2, n-2)$ in the proof of Lemma 6.20.

a series of claims that we may assume that p_2 contains $(m-2, n-2)$. In particular, we show that if p_2 does not contain $(m-2, n-2)$, then there is another path in M_2 that does contain $(m-2, n-2)$ and dominates p_2 .

Claim 6.20.2. *We may assume that there is a unique $j \in [n-2]$ such that p_2 contains $(m-1, j)$.*

Proof. Clearly, p_2 has to pass through the row $m-1$ at some point. We show that we may assume that there is a unique such point. Suppose not and let j_1, \dots, j_t be such that p_2 contains all $(m-1, j_i)$, where $i \in [t]$. By the definition of a path in a matrix, we have that $j_{i+1} = j_i + 1$ for all $i \in [t-1]$. Let p'_2 be the path obtained from p_2 by replacing, for each $i \in [t-1]$, the element $(m-1, j_i)$ with the element $(m-2, j_i)$. Since $r(m-2) \leq r(m-1)$ (recall that $m-1 \in \text{argmax}(r)$), it is not difficult to see that p'_2 dominates p_2 , and clearly, p'_2 satisfies the condition of the claim. \square

Claim 6.20.3. *Let $j \in [n-3]$ be such that p_2 contains $(m-1, j)$. If $j = n-2k+1$ for some $k \in \mathbb{N}$ with $2k+1 \leq n-1$, then there is a path p'_2 that dominates p_2 and contains $(m-1, j+1)$.*

Proof. For an illustration see Figure 6.8a. First, by Claim 6.20.2, we may assume that j is unique. Moreover, since $j = n-2k+1$ and $j+1 = n-2k+2 = n-2(k-1)$, we have that $c(j) = \max_k(c)$ and $c(j+1) = \min_k(c)$, respectively, and therefore $c(j+1) \leq c(j)$. Hence, we may assume that the element after $(m-1, j)$ in p_2 is $(m, j+1)$: if p_2 contained (m, j) we could simply remove (m, j) from p_2 without changing the fact that p_2 is a dominating path since $M[m, j] > M[m, j+1]$. We modify p_2 as follows. We remove $(m-1, j)$, and add $(m-2, j)$ (if not already present), followed by $(m-2, j+1)$ and then $(m-1, j+1)$. For each $x \in \{M[m -$

$2, j], M[m - 2, j + 1], M[m - 1, j + 1]\}$, we have that $x < M[m - 1, j]$ (recall that $r(m - 2) < r(m - 1)$ and $c(j + 1) < c(j)$). Hence, the resulting path dominates p_2 and it contains $(m - 1, j + 1)$. \square

Claim 6.20.4. *Let $j \in [n - 4]$ be such that p_2 contains $(m - 1, j)$. If $j = n - 2(k - 1)$ for some $k \in [3.. \lfloor \frac{n}{2} \rfloor]$, then there is a path p'_2 that dominates p_2 and contains $(m - 1, j + 2)$.*

Proof. For an illustration see Figure 6.8b. Again, by Claim 6.20.2, we may assume that j is unique. Since $j = n - 2(k - 1)$, we have that $c(j) = \min_k(c)$. First, if not already present, we insert $(m - 2, j)$ just before $(m - 1, j)$ in p_2 . This does not change the fact that p_2 is a dominating path, since $M[m - 2, j] < M[m - 1, j]$ (recall that $r(m - 2) < r(m - 1)$). Next, consider the 3×3 submatrix $L := M[(m - 2)..m, j..(j + 2)]$. Note that L is the submatrix of M restricted to the rows $\min(r)$, $\max(r)$, and $\min_2(r)$, and the columns $\min_k(c)$, $\max_{k-1}(c)$, and $\min_{k-1}(c)$. Furthermore, we have that p_2 restricted to L is equal to $p_r(L)$. We show that $p_\perp(L)$ dominates $p_r(L)$, from which we can conclude that we can obtain a path p'_2 from p_2 that contains $(m - 1, j + 2)$ and dominates p_2 by replacing $p_r(L)$ with $p_\perp(L)$. By Lemma 6.18, it suffices to show that $M[m - 2, j + 1] \leq M[m - 1, j]$, in other words, that $\max_{k-1}(c) + \min_2(r) \leq \max(r) + \min_k(c)$.

By the assumption of the lemma, we have that $M[m - 2, n - 1] \leq M[m - 1, n - 2]$, hence,

$$\begin{aligned} \max(c) + \min_2(r) &\leq \max(r) + \min_2(c), \text{ and so:} \\ \max(c) - \min_2(c) &\leq \max(r) - \min_2(r). \end{aligned}$$

Next, we have that for all $j \in [\lfloor n/2 \rfloor]$,

$$\max(c) - \min_2(c) \geq \max_j(c) - \min_{j+1}(c).$$

Putting the two together, we have that

$$\begin{aligned} \max_{k-1}(c) - \min_k(c) &\leq \max(r) - \min_2(r), \text{ and so:} \\ \max_{k-1}(c) + \min_2(r) &\leq \max(r) + \min_k(c), \end{aligned}$$

which concludes the proof of the claim. \square

We are now ready to conclude the proof.

Claim 6.20.5. $p_1^+ \prec p_2^+$.

Proof. By repeated application of Claims 6.20.3 and 6.20.4, we know that there is a path p'_2 in M_2 that contains $(m - 1, n - 2)$. Furthermore, we may assume that p'_2 contains $(m - 2, n - 2)$ as well: we can simply add this element if it is not already present; since $M[m - 2, n - 2] \leq M[m - 1, n - 2]$, this does not change the property

that $p'_2 \prec p_2$. Now, let p''_2 be the subpath of p'_2 ending in $(m-2, n-2)$. (Note that $p''_2 \circ (m-2, n-1), (m-2, n) \in \mathcal{P}(M_1)$.) Then,

$$p_1^+ \prec p''_2 \circ (m-2, n-1), (m-2, n), (m-1, n), (m, n) \quad (6.4)$$

$$\prec p'_2 \circ (m, n-1), (m, n) \quad (6.5)$$

$$\prec p_2^+, \quad (6.6)$$

where (6.4) is due to $p_1 \prec \mathcal{P}(M_1)$ and therefore $p_1 \prec p''_2 \circ (m-2, n-1), (m-2, n)$. Next, (6.5) follows from an application of Lemma 6.18 to the 3×3 -submatrix $M[(m-2)..m, (n-2)..n]$ and (6.6) is guaranteed since $p'_2 \prec p_2$. \square

This concludes the proof of (i) and (ii) can be shown symmetrically. \square

As the previous lemma always assumes that $m \geq 3$ and $n \geq 3$, we observe the corresponding base case which occurs when either $m \leq 2$ or $n \leq 2$. This base case is justified by the observation that in the bottom case, the last row and column of M are minimum.

Observation 6.21 (Base Case - Bottom). Let r and c be typical sequences of length m and n , respectively, and let M be the merge matrix of r and c . Suppose that $m \in \text{argmin}(r)$ and $n \in \text{argmin}(c)$. If $m \leq 2$ ($n \leq 2$), then³

$$p^* := (1, 1), (m, 1), (m, 2), \dots, (m, n) \quad (p^* := (1, 1), (1, n), (2, n), \dots, (m, n))$$

dominates $\mathcal{P}(M)$, i.e. $p^* \prec \mathcal{P}(M)$.

By symmetry, we have the following consequence of Lemma 6.20.

Corollary 6.22 (Chop Lemma - Top). Let r and c be typical sequences of length $m \geq 3$ and $n \geq 3$, respectively, and let M be the merge matrix of r and c . Suppose that $1 \in \text{argmin}(r)$ and $1 \in \text{argmin}(c)$ and let $M_1 := M[3..m, 1..n]$ and $M_2 := M[1..m, 3..n]$ and for all $h \in [2]$, let $p_h \prec \mathcal{P}(M_h)$. Let $p_1^+ := (1, 1), (2, 1) \circ p_1$ and $p_2^+ := (1, 1), (1, 2) \circ p_2$.

(i) If $M[3, 2] \leq M[2, 3]$, then $p_1^+ \prec \mathcal{P}(M)$.

(ii) If $M[2, 3] \leq M[3, 2]$, then $p_2^+ \prec \mathcal{P}(M)$.

Again, we observe the corresponding base case.

Observation 6.23 (Base Case - Top). Let r and c be typical sequences of length m and n , respectively, and let M be the merge matrix of r and c . Suppose that $1 \in \text{argmin}(r)$ and $1 \in \text{argmin}(c)$. If $m \leq 2$ ($n \leq 2$), then

$$p^* := (1, 1), (1, 2), \dots, (1, n), (m, n) \quad (p^* := (1, 1), (2, 1), \dots, (m, 1), (m, n))$$

dominates $\mathcal{P}(M)$, i.e. $p^* \prec \mathcal{P}(M)$.

³Note that in the following equation, if $m = 1$, then strictly speaking we would have that p^* repeats the element $(1, 1)$ twice which is of course not our intention. For the sake of a clear presentation though, we will ignore this slight abuse of notation, also in similar instances throughout this section.

```

Input : Typical sequences  $r(1), \dots, r(m)$  and  $c(1), \dots, c(n)$ 
Output: A dominating merge of  $r$  and  $c$ 
1 Let  $i \in \text{argmin}(r)$  and  $j \in \text{argmin}(c)$ ;
2 return Chop-bottom ( $r[1..i]$ ,  $c[1..j]$ )  $\circ$  Chop-top ( $r[i..m]$ ,  $c[j..n]$ );
3 Procedure Chop-bottom( $r$  and  $c$  as above)
4   if  $m \leq 2$  then return  $r(1) + c(1)$ ,  $r(m) + c(1)$ ,  $r(m) + c(2)$ ,  $\dots$ ,
     $r(m) + c(n)$ ;
5   if  $n \leq 2$  then return  $r(1) + c(1)$ ,  $r(1) + c(n)$ ,  $r(2) + c(n)$ ,  $\dots$ ,
     $r(m) + c(n)$ ;
6   if  $r(m-2) + c(n-1) \leq r(m-1) + c(n-2)$  then return
    Chop-bottom( $r[1..(m-2)]$ ,  $c$ )  $\circ$  ( $r(m-1) + c(n)$ ),  $r(m) + c(n)$ ;
7   if  $r(m-1) + c(n-2) \leq r(m-2) + c(n-1)$  then return
    Chop-bottom( $r, c[1..(n-2)]$ )  $\circ$  ( $r(m) + c(n-1)$ ),  $r(m) + c(n)$ ;
8 Procedure Chop-top( $r$  and  $c$  as above)
9   if  $m \leq 2$  then return  $r(1) + c(1)$ ,  $r(1) + c(2)$ ,  $\dots$ ,  $r(1) + c(n)$ ,
     $r(m) + c(n)$ ;
10  if  $n \leq 2$  then return  $r(1) + c(1)$ ,  $r(2) + c(1)$ ,  $\dots$ ,  $r(m) + c(1)$ ,
     $r(m) + c(n)$ ;
11  if  $r(3) + c(2) \leq r(2) + c(3)$  then return
     $r(1) + c(1)$ ,  $(r(2) + c(1)) \circ$  Chop-top( $r[3..m]$ ,  $c$ );
12  if  $r(2) + c(3) \leq r(3) + c(2)$  then return
     $r(1) + c(1)$ ,  $(r(1) + c(2)) \circ$  Chop-top( $r, c[3..n]$ );

```

Algorithm 4: The Split-and-Chop Algorithm

6.2.4 The Split-and-Chop Algorithm

Equipped with the Split Lemma and the Chop Lemmas, we are now ready to give the algorithm that computes a dominating merge of two typical sequences. Consequently, we call this algorithm the ‘Split-and-Chop Algorithm’.

Lemma 6.24. *Let r and c be typical sequences of length m and n , respectively. Then, there is an algorithm that finds in $\mathcal{O}(m+n)$ time a dominating path in the merge matrix of r and c .*

Proof. The algorithm practically derives itself from the Split Lemma (Lemma 6.17) and the Chop Lemmas (Lemma 6.20 and Corollary 6.22). However, to make the algorithm run in the claimed time bound, we are not able to construct the merge matrix of r and c . This turns out to be not necessary, as we can simply read off the crucial values upon which the recursion of the algorithm depends from the sequences directly. The details are given in Algorithm 4.

The runtime of the Chop-subroutines can be computed as $T(m+n) \leq T(m+n-2) + \mathcal{O}(1)$, which resolves to $\mathcal{O}(m+n)$. Correctness follows from Lemmas 6.17 and 6.20 and Corollary 6.22 with the base cases given in Observations 6.21 and 6.23. \square

6.2.5 Generalization to Arbitrary Integer Sequences

In this section we show how to generalize Lemma 6.24 to arbitrary integer sequences. In particular, we will show how to construct from a merge of two typical sequences $\tau(r)$ and $\tau(s)$ that dominates all of their merges, a merge of r and s that dominates all merges of r and s . The claimed result then follows from an application of Lemma 6.24. We illustrate the following construction in Figure 6.9.

The Typical Lift. Let r and s be integer sequences and let $t \in \tau(r) \oplus \tau(s)$. Then, the *typical lift* of t , denoted by $\rho(t)$, is an integer sequence $\rho(t) \in r \oplus s$, obtained from t as follows. For convenience, we will consider $\rho(t)$ as a path in the merge matrix M of r and s .

Step 1. We construct $t' \in \tau(r) \boxplus \tau(s)$ such that $t' \prec t$ using Lemma 6.15. Throughout the following, consider t' to be a path in the merge matrix M_τ of $\tau(r)$ and $\tau(s)$.

Step 2. First, we initialize $\rho_t^1 := t'(1) = (1, 1)$. For $i = \{2, \dots, l(t')\}$, we proceed inductively as follows. Let $(i_r, i_s) = t(i)$ and let $(i'_r, i'_s) = t(i-1)$. (Note that $t(i-1)$ and $t(i)$ are indices in M_τ .) Let furthermore (j_r, j_s) be the index in M corresponding to (i_r, i_s) , and let (j'_r, j'_s) be the index in M corresponding to (i'_r, i'_s) . Assume by induction that $\rho_t^{i-1} \in \mathcal{P}(M[1..j'_r, 1..j'_s])$. We show how to extend ρ_t^{i-1} to a path in ρ_t^i in $M[1..j_r, 1..j_s]$. Since t' is non-diagonal, we have that $(i'_r, i'_s) \in \{(i_r - 1, i_s), (i_r, i_s - 1)\}$, so one of the two following cases applies.

Case S2.1 ($i'_r = i_r - 1$ and $i'_s = i_s$). In this case, we let

$$\rho_t^i := \rho_t^{i-1} \circ (j'_r + 1, j_s), \dots, (j_r, j_s).$$

Case S2.2 ($i'_r = i_r$ and $i'_s = i_s - 1$). In this case, we let

$$\rho_t^i := \rho_t^{i-1} \circ (j_r, j'_s + 1), \dots, (j_r, j_s).$$

Step 3. We return $\rho(t) := \rho_t^{l(t')}$.

Furthermore, it is readily seen that the typical lift contains no diagonal steps: we obtain it from a non-diagonal path in the merge matrix of $\tau(r)$ and $\tau(s)$ by inserting vertical and horizontal paths from the merge matrix of r and s between consecutive elements. Moreover, it is computable in linear time, with Step 1 taking linear time by Lemma 6.15. We summarize in the following observation.

Observation 6.25. Let r and s be integer sequences of length m and n , respectively, and let $t \in \tau(r) \oplus \tau(s)$. Then, $\rho(t) \in r \boxplus s$, and $\rho(t)$ can be computed in time $\mathcal{O}(m + n)$.

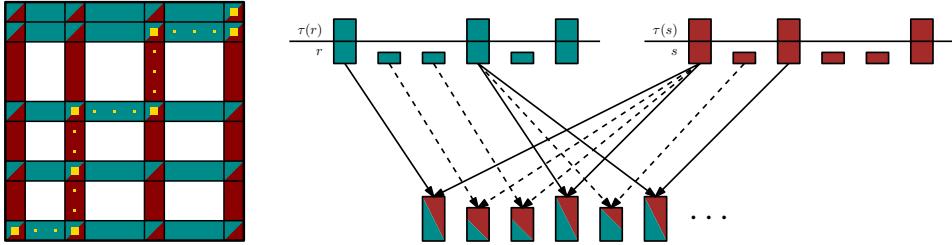


Figure 6.9: Illustration of the typical lift. On the left side, the view of the merge matrix M , with the rows and columns corresponding to elements of the typical sequences highlighted. Inside there, M_τ can be seen as a highlighted submatrix. The merge t' is depicted as the large yellow squares within M_τ and the small yellow squares outside of M_τ show its completion to the typical lift of t . On the right side, an illustration that does not rely on the ‘matrix view’.

We now show that if $t \in \tau(r) \oplus \tau(s)$ dominates all merges of $\tau(r)$ and $\tau(s)$, then the typical lift of t dominates all merges of r and s .

Lemma 6.26. *Let r and s be integer sequences and let $q \in r \oplus s$. Let $t \in \tau(r) \oplus \tau(s)$ such that $t \prec \tau(r) \oplus \tau(s)$. Then, $\rho(t) \prec q$.*

Proof. Let $t' \in \tau(r) \boxplus \tau(s)$ be the non-diagonal merge such that $t' \prec t$ used in the construction of $\rho(t)$. We argue that $\rho(t) \prec t'$. To see this, let M be the merge matrix of r and s and consider any (j'_r, j'_s) and (j_r, j_s) as in Step 2, and suppose that $j'_s = j_s$. (Note that either $j'_s = j_s$ or $j'_r = j_r$.) As the only elements of the typical sequence of r in $[j'_r..j_r]$ are $r(j'_r)$ and $r(j_r)$, we know that either for all $h_r \in [j'_r..j_r]$, $r(j'_r) \leq r(h_r) \leq r(j_r)$, or for all $h_r \in [j'_r..j_r]$, $r(j'_r) \geq r(h_r) \geq r(j_r)$. Therefore, in an extension of t' , we can repeat the index that yields $\max\{M[j'_r, j_s], M[j_r, j_s]\}$ sufficiently many (i.e. $j_r - j'_r$) times to ensure that the value of the extension of t' is an upper bound for all values of $\rho(t)$ in these positions.

To finish the proof, we have by Lemma 6.10(iii) that there exists a $q' \in \tau(r) \oplus \tau(s)$ such that $q' \prec q$. Since $t \prec \tau(r) \oplus \tau(s)$, we can conclude:

$$\rho(t) \prec t' \prec t \prec q' \prec q. \quad \square$$

We wrap up and prove the Merge Dominator Lemma (Lemma 6.11), stated here in the slightly stronger form that the dominating merge is non-diagonal (which is necessary for the applications in chapter 7).

Lemma 6.27 (Merge Dominator Lemma). *Let r and c be integer sequence of length m and n , respectively. There exists a dominating non-diagonal merge of r and c , i.e. an integer sequence $t \in r \boxplus c$ such that $t \prec r \oplus c$, and this dominating merge can be computed in time $\mathcal{O}(m + n)$.*

Proof. The algorithm proceeds in the following steps.

Step 1. Compute $\tau(r)$ and $\tau(c)$.

Step 2. Apply the Split-and-Chop Algorithm on input $(\tau(r), \tau(c))$ to obtain $t \prec \tau(r) \oplus \tau(c)$.

Step 3. Return the typical lift $\rho(t)$ of t .

Correctness of the above algorithm follows from Corollary 6.14 and Lemmas 6.24 and 6.26 which together guarantee that $\rho(t) \prec r \oplus c$, and by Observation 6.25, $\rho(t)$ is a non-diagonal merge, i.e. $\rho(t) \in r \boxplus c$. By Lemma 6.9, Step 1 can be done in time $\mathcal{O}(m + n)$, by Lemma 6.24, Step 2 takes time $\mathcal{O}(m + n)$ as well, and by Observation 6.25, the typical lift of t can also be computed in time $\mathcal{O}(m + n)$. Hence, the overall runtime of the algorithm is $\mathcal{O}(m + n)$. \square

6.3 Employing the Merge Dominator Lemma

As discussed in the introduction to this chapter, the Merge Dominator Lemma on its own is insufficient to give an asymptotic improvement in the running time of Bodlaender's FPT-algorithm [29] for computing the treewidth of a graph. However, this does not rule out that it provides a speed-up in a practical implementation of the algorithm. An experimental evaluation due to Röhrig [257] showed early on that the algorithm [29] is not practically feasible even for very small values of the treewidth of the input graphs.

The Merge Dominator Lemma can prevent a blow-up in the table sizes at join nodes in the Bodlaender-Kloks subroutine [37] for treewidth- k . Given a graph of slightly too high treewidth, this subroutine decides whether the treewidth (or pathwidth) of the input graph is at most k . In a first study of the practical implications of the Merge Dominator Lemma, one could test separately how it influences the performance of this subroutine alone.

Width Measures of Series Parallel Digraphs

In this chapter, we use the Merge Dominator Lemma to compute two width measures of series parallel digraphs, a subclass of directed acyclic graphs (DAGs), formally defined below. Specifically, we show that the (WEIGHTED) CUTWIDTH and MODIFIED CUTWIDTH problems on DAGs, which given a directed acyclic graph on n vertices, ask for the topological order that minimizes the *cutwidth* and *modified cutwidth*, respectively, can be solved in $\mathcal{O}(n^2)$ time on series parallel digraphs. Note that the restriction of the solution to be a *topological* order has been made as well in other works, e.g. [35].

Our algorithm for CUTWIDTH of series parallel digraphs has the same structure as the dynamic programming algorithm for undirected CUTWIDTH [35], but, in addition to obeying directions of edges, we have a step that only keeps characteristics that are not dominated by another characteristic in a table of characteristics. Now, with help of our Merge Dominator Lemma, we can show that in the case of series parallel digraphs, there is a unique dominating characteristic; the dynamic programming algorithm reverts to computing for each intermediate graph a single ‘optimal partial solution’. This strategy also works in the presence of edge weights, which gives the algorithm for the corresponding WEIGHTED CUTWIDTH problem on series parallel digraphs. Note that the cutwidth of a directed acyclic graph is at least the maximum indegree or outdegree of a vertex; e.g., a series parallel digraph formed by the parallel composition of $n - 2$ paths with three vertices has n vertices and cutwidth $n - 2$. To compute the *modified* cutwidth of a series parallel digraph, we give a linear-time reduction to the WEIGHTED CUTWIDTH problem on series parallel digraphs.

We now introduce series parallel digraphs. Note that the following definition coincides with the notion of ‘edge series-parallel multidigraphs’ in [277]. For an illustration see Figure 7.1.

Definition 7.1 (Series Parallel Digraph (SPD)). A (multi-)digraph G with an ordered pair of *terminals* $(s, t) \in V(G) \times V(G)$ is called *series parallel digraph (SPD)*,

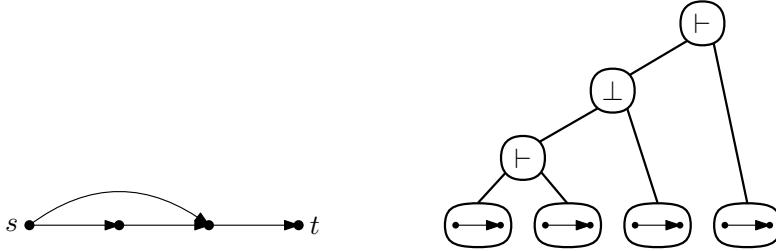


Figure 7.1: A series parallel digraph G on the left, and a decomposition tree that yields G on the right.

often denoted by $(G, (s, t))$, if one of the following hold.

- (i) $(G, (s, t))$ is a single arc directed from s to t , i.e. $V(G) = \{s, t\}$, $A(G) = \{(s, t)\}$.
- (ii) $(G, (s, t))$ can be obtained from two series parallel digraphs $(G_1, (s_1, t_1))$ and $(G_2, (s_2, t_2))$ by one of the following operations.
 - (a) *Series Composition.* $(G, (s, t))$ is obtained by taking the disjoint union of G_1 and G_2 , identifying t_1 and s_2 , and letting $s = s_1$ and $t = t_2$. In this case we write $(G, (s, t)) = (G_1, (s_1, t_1)) \vdash (G_2, (s_2, t_2))$ or simply $G = G_1 \vdash G_2$.
 - (b) *Parallel Composition.* $(G, (s, t))$ is obtained by taking the disjoint union of G_1 and G_2 , identifying s_1 and s_2 , and identifying t_1 and t_2 , and letting $s = s_1 = s_2$ and $t = t_1 = t_2$. In this case we write $(G, (s, t)) = (G_1, (s_1, t_1)) \perp (G_2, (s_2, t_2))$, or simply $G = G_1 \perp G_2$.

It is not difficult to see that each series parallel digraph is acyclic. One can naturally associate a notion of *decomposition trees* with series parallel digraphs as follows. A decomposition tree T is a rooted and ordered binary tree whose leaves are labeled with a single arc, and each internal node $t \in V(T)$ with left child ℓ and right child r is either a *series node* or a *parallel node*. We then associate an SPD G_t with each node $t \in V(T)$. If t is a leaf, then G_t is a single arc oriented from one terminal to the other. If t is an internal node, then G_t is $G_\ell \vdash G_r$ if t is a series node and $G_\ell \perp G_r$ if t is a parallel node. It is clear that for each SPD G , there is a decomposition tree T with root τ such that $G = G_\tau$. In that case we say that T *yields* G . Valdes et al. [277] have shown that one can decide in linear time whether a directed graph G is an SPD and if so, find a decomposition tree that yields G .

Theorem 7.2 (Valdes et al. [277]). *Let G be a directed graph on n vertices and m arcs. There is an algorithm that decides in time $\mathcal{O}(n + m)$ whether G is a series parallel digraph and if so, it outputs a decomposition tree that yields G .*

We consider the following width measures of DAGs. Let G be a DAG on n vertices. A *topological order* of G is a linear order $\pi: V(G) \rightarrow [n]$ such that for all

arcs $uv \in A(G)$, we have that $\pi(u) < \pi(v)$. We denote the set of all topological orders of G by $\Pi(G)$. Note that we restrict the orderings of the vertices that we consider to *topological* orderings, as it is done e.g. in [35].

Definition 7.3. Let G be a directed acyclic graph and let $\pi \in \Pi(G)$ be a topological order of G .

- (i) The *cutwidth* of π is $\text{cutw}(\pi) := \max_{i \in [n-1]} |\{uv \in A(G) \mid \pi(u) \leq i \wedge \pi(v) > i\}|$.
- (ii) The *modified cutwidth* of π is

$$\text{mcutw}(\pi) := \max_{i \in [n]} |\{uv \in A(G) \mid \pi(u) < i \wedge \pi(v) > i\}|.$$

We define the cutwidth and modified cutwidth of a directed acyclic graph G as the minimum of the respective measure over all topological orders of G , i.e.

$$\text{cutw}(G) := \min_{\pi \in \Pi(G)} \text{cutw}(\pi) \text{ and } \text{mcutw}(G) := \min_{\pi \in \Pi(G)} \text{mcutw}(\pi)$$

A more intuitive way of viewing cutwidth and modified cutwidth is as follows. Let G be a DAG on n vertices and $\pi \in \Pi(G)$ one of its topological orders. Let v_1, \dots, v_n denote the vertices of G ordered according to π . Then, the cutwidth of π is the maximum, over all $i \in [n-1]$ of the number of arcs that have their tail vertex among $\{v_1, \dots, v_i\}$ and their head vertex among $\{v_{i+1}, \dots, v_n\}$. The difference to the *modified cutwidth* is that for each $i \in [n]$, we take the maximum of the number of arcs with tail in $\{v_1, \dots, v_{i-1}\}$ and head in $\{v_{i+1}, \dots, v_n\}$.

Therefore, the cutwidth considers cuts *between positions i and $i + 1$* while the modified cutwidth considers cuts *at each position i* .

7.1 Cutwidth of SPDs

We now deal with the following computational problem.

CUTWIDTH OF SERIES PARALLEL DIGRAPHS

<i>Input:</i>	A series parallel digraph G .
<i>Question:</i>	What is the cutwidth of G ?

Given a series parallel digraph G , we follow a bottom-up dynamic programming scheme along the decomposition tree T that yields G . Each node $t \in V(T)$ has a subgraph G_t of G associated with it, that is also series parallel. Naturally, we use the property that G_t is obtained either via series or parallel composition of the SPD's associated with its two children.

To make this problem amenable to be solved using merges of integer sequences, we define the following notion of a cut-size sequence of a topological order of a directed acyclic graph which records for each position in the order, how many arcs cross it.

Definition 7.4 (Cut-Size Sequence). Let G be a directed acyclic graph on n vertices and let $\pi \in \Pi(G)$ be a topological order of G . The sequence $x(1), \dots, x(n-1)$, where for $i \in [n-1]$,

$$x(i) = |\{uv \in A(G) \mid \pi(u) \leq i \wedge \pi(v) > i\}|,$$

is the *cut-size sequence* of π , and denoted by $\sigma(\pi)$. For a set of topological orders $\Pi' \subseteq \Pi(G)$, we let $\sigma(\Pi') := \{\sigma(\pi) \mid \pi \in \Pi'\}$.

Throughout the remainder of this section, we slightly abuse notation: If G_1 and G_2 are SPD's that are being composed with a series composition, and $\pi_1 \in \Pi(G_1)$ and $\pi_2 \in \Pi(G_2)$, then we consider $\pi = \pi_1 \circ \pi_2$ to be the concatenation of the two topological orders where $t_2 = s_1$ only appears *once* in π .

We first argue via two simple observations that when computing the cutwidth of a series parallel digraph G by following its decomposition tree in a bottom up manner, we only have to keep track of a set of topological orders that induce a set of cut-size sequences that dominate all cut-size sequences of G .

Observation 7.5. Let G be a DAG and $\pi, \lambda \in \Pi(G)$. If $\sigma(\pi) \prec \sigma(\lambda)$, then $\text{cutw}(\pi) \leq \text{cutw}(\lambda)$.

This is simply due to the fact that $\sigma(\pi) \prec \sigma(\lambda)$ implies that $\max(\sigma(\pi)) \leq \max(\sigma(\lambda))$. Next, if G is obtained from G_1 and G_2 via series or parallel composition, and we have $\pi_1, \lambda_1 \in \Pi(G_1)$ such that $\sigma(\pi_1) \prec \sigma(\lambda_1)$, then it is always beneficial to choose π_1 over λ_1 , and π_1 can be disregarded.

Observation 7.6. Let G be an SPD that is obtained via series or parallel composition from SPD's G_1 and G_2 . Let $\pi_1, \lambda_1 \in \Pi(G_1)$ be such that $\sigma(\pi_1) \prec \sigma(\lambda_1)$. Let $\pi, \lambda \in \Pi(G)$ be such that $\pi|_{V(G_1)} = \pi_1$, $\lambda|_{V(G_1)} = \lambda_1$, and for all $v \in V(G_2)$, $\pi(v) = \lambda(v)$. Then, $\sigma(\pi) \prec \sigma(\lambda)$.

The previous observation is justified as follows. Let $\sigma(\pi) = x(1), \dots, x(n-1)$ and $\sigma(\lambda) = y(1), \dots, y(n-1)$. Then, for each $i \in [n-1]$, the arcs of G_2 contribute equally to the values $x(i)$ and $y(i)$ (in particular since G_1 and G_2 are arc-disjoint). Therefore, we can use extensions of $\sigma(\pi_1)$ and $\sigma(\lambda_1)$ that witnesses that $\sigma(\pi_1) \prec \sigma(\lambda_1)$ to construct extensions of $\sigma(\pi)$ and $\sigma(\lambda)$ that witness that $\sigma(\pi) \prec \sigma(\lambda)$.

The following lemma states that the cut-size sequences of an SPD G can be computed by pairwise concatenation or non-diagonal merging (depending on whether G is obtained via series or parallel composition) of the two smaller SPD's that G is obtained from. Intuitively speaking, the reason why we can only consider *non-diagonal* merges is the following. When G is obtained from G_1 and G_2 via parallel composition, then each topological order of G can be considered the ‘merge’ of a topological order of G_1 and one of G_2 , where each position (apart from the first and the last) contains a vertex *either* from G_1 *or* from G_2 . Now, in a merge of a cut-size sequence of G_1 with a cut-size sequence of G_2 , a diagonal step would essentially mean

that in some position, we insert both a vertex from G_1 and a vertex of G_2 ; this is of course not possible.

Lemma 7.7. *Let G_1 and G_2 be SPD's. Then the following hold.*

$$(i) \sigma(\Pi(G_1 \vdash G_2)) = \sigma(\Pi(G_1)) \odot \sigma(\Pi(G_2)).$$

$$(ii) \sigma(\Pi(G_1 \perp G_2)) = \sigma(\Pi(G_1)) \boxplus \sigma(\Pi(G_2)).$$

Proof. (i). Let $\sigma(\pi) \in \sigma(\Pi(G_1 \vdash G_2))$ be such that π is a topological order of $G_1 \vdash G_2$. Then, π consists of two contiguous parts, namely $\pi_1 := \pi|_{V(G_1)} \in \Pi(G_1)$ followed by $\pi_2 := \pi|_{V(G_2)} \in \Pi(G_2)$. Since there are no arcs from $V(G_1) \setminus \{t_1\}$ to $V(G_2) \setminus \{s_2\}$, we have that $\sigma(\pi) = \sigma(\pi_1) \circ \sigma(\pi_2) \in \sigma(\Pi(G_1)) \odot \sigma(\Pi(G_2))$. The other inclusion follows similarly.

(ii). Let $\sigma(\pi) \in \sigma(\Pi(G_1 \perp G_2))$ be such that π is a topological order of $G_1 \perp G_2$. Let $\pi_1 := \pi|_{V(G_1)}$ and $\pi_2 := \pi|_{V(G_2)}$. It is clear that $\pi_1 \in \Pi(G_1)$ and that $\pi_2 \in \Pi(G_2)$. Let $\sigma(\pi) = x(1), \dots, x(n-1)$, $\sigma(\pi_1) = y_1(1), \dots, y_1(n_1 - 1)$, and $\sigma(\pi_2) = y_2(1), \dots, y_2(n_2 - 1)$. For any $i \in \{1, \dots, n-1\}$, let i_1 be the maximum index such that $\pi(\pi_1^{-1}(i_1)) \leq i$, and define i_2 accordingly. Then, the set of arcs that cross the cut between positions i and $i+1$ in π is the union of the set of arcs crossing the cut between positions i_1 and $i_1 + 1$ in π_1 and the set of arcs crossing the cut between positions i_2 and $i_2 + 1$ in π_2 . Since G_1 and G_2 are arc-disjoint, this means that $x(i) = y_1(i_1) + y_2(i_2)$. Together with the observation that each vertex at position $i+1 < n$ in π is either from G_1 or from G_2 , we have that

$$x(i+1) \in \{y_1(i_1 + 1) + y_2(i_2), y_1(i_1) + y_2(i_2 + 1)\},$$

in other words, we have that $\sigma(\pi) \in \sigma(\pi_1) \boxplus \sigma(\pi_2) \subseteq \sigma(\Pi(G_1)) \boxplus \sigma(\Pi(G_2))$. The other inclusion can be shown similarly, essentially using the fact that we are only considering non-diagonal merges. \square

We now prove the crucial lemma of this section which states that we can compute a dominating cut-size sequence of an SPD G from dominating cut-size sequences of the smaller SPD's that G is obtained from. For technical reasons, we assume in the following lemma that G has no parallel arcs, which does not affect the algorithm presented in this section.

Lemma 7.8. *Let G be an SPD without parallel arcs. Then there is a topological order π^* of G such that $\sigma(\pi^*)$ dominates all cut-size sequences of G . Moreover, the following hold. Let G_1 and G_2 be SPD's and for $r \in [2]$, let π_r^* be a topological order of G_r such that $\sigma(\pi_r^*)$ dominates all cut-size sequences of G_r .*

(i) *If $G = G_1 \vdash G_2$, then $\pi^* = \pi_1^* \circ \pi_2^*$.*

(ii) *If $G = G_1 \perp G_2$, then π^* can be found as the topological order of G such that $\sigma(\pi^*)$ dominates $\sigma(\pi_1^*) \boxplus \sigma(\pi_2^*)$.*

Proof. We prove the lemma by induction on the number of vertices of G . If $|V(G)| = 2$, then the claim is trivially true (there is only one topological order). Suppose that $|V(G)| =: n > 2$. Since $n > 2$ and G has no parallel arcs, we know that G can be obtained from two SPD's G_1 and G_2 via series or parallel composition with $|V(G_1)| =: n_1 < n$ and $|V(G_2)| =: n_2 < n$. By the induction hypothesis, for $r \in [2]$, there is a unique topological order π_r^* such that $\sigma(\pi_r^*)$ dominates all cut-size sequences of G_r .

Suppose $G = G_1 \vdash G_2$. Since $\sigma(\pi_1^*)$ dominates all cut-size sequences of G_1 and $\sigma(\pi_2^*)$ dominates all cut-size sequences of G_2 , we can conclude using Lemma 6.10(v) that $\sigma(\pi_1^*) \circ \sigma(\pi_2^*)$ dominates $\sigma(\Pi(G_1)) \odot \sigma(\Pi(G_2))$ which together with Lemma 7.7(i) allows us to conclude that $\sigma(\pi_1^*) \circ \sigma(\pi_2^*) = \sigma(\pi_1^* \circ \pi_2^*)$ dominates all cut-size sequences of G . This proves (i).

Suppose that $G = G_1 \perp G_2$, and let π^* be a topological order of G such that $\sigma(\pi^*)$ dominates $\sigma(\pi_1^*) \boxplus \sigma(\pi_2^*)$. We show that $\sigma(\pi^*)$ dominates $\sigma(\Pi(G))$. Let $\pi \in \Pi(G)$. By Lemma 7.7(ii), there exist topological orders $\pi_1 \in \Pi(G_1)$ and $\pi_2 \in \Pi(G_2)$ such that $\sigma(\pi) \in \sigma(\pi_1) \boxplus \sigma(\pi_2)$. In other words, there are extensions e_1 of $\sigma(\pi_1)$ and e_2 of $\sigma(\pi_2)$ of the same length such that $\sigma(\pi) = e_1 + e_2$. For $r \in [2]$, since $\sigma(\pi_r^*) \prec \sigma(\pi_r)$, we have that $\sigma(\pi_r^*) \prec e_r$. By Lemma 6.10(ii),¹ there exists some $f \in \sigma(\pi_1^*) \oplus \sigma(\pi_2^*)$ such that $f \prec e_1 + e_2$, and by Lemma 6.15, there is some $f' \in \sigma(\pi_1^*) \boxplus \sigma(\pi_2^*)$ such that $f' \prec f$. Since $\sigma(\pi^*) \prec \sigma(\pi_1^*) \boxplus \sigma(\pi_2^*)$, we have that $\sigma(\pi^*) \prec f'$, and hence (ii) follows:

$$\sigma(\pi^*) \prec f' \prec f \prec e_1 + e_2 = \sigma(\pi). \quad \square$$

We are now ready to prove the first main result of this chapter.

Theorem 7.9. *Let G be an SPD on n vertices. There is an algorithm that computes in time $\mathcal{O}(n^2)$ the cutwidth of G , and outputs a topological ordering that achieves the upper bound.*

Proof. We may assume that G has no parallel arcs; if so, we simply subdivide all but one of the parallel arcs. This neither changes the cutwidth, nor the fact that G is series parallel. We can therefore apply Lemma 7.8 on G in the correctness proof later.

We use the algorithm of Valdes et al. [277] to compute in time $\mathcal{O}(n + |A(G)|)$ a decomposition tree T that yields G , see Theorem 7.2. We process T in a bottom-up fashion, and at each node $t \in V(T)$, compute a topological order π_t of G_t , the series parallel digraph associated with node t , such that $\sigma(\pi_t)$ dominates all cut-size sequences of G_t . Let $t \in V(T)$.

Case 1 (t is a leaf node). In this case, G_t is a single arc and there is precisely one topological order of G_t ; we return that order.

¹Take $r = e_1$, $s = e_2$, $r_0 = \sigma(\pi_1)$, and $s_0 = \sigma(\pi_2)$.

Case 2 (t is a series node with left child ℓ and right child r). In this case, we look up π_ℓ , a topological order such that $\sigma(\pi_\ell)$ dominates all cut-size sequences of G_ℓ , and π_r , a topological order such that $\sigma(\pi_r)$ dominates all cut-size sequences of G_r . Following Lemma 7.8(i), we return $\pi_\ell \circ \pi_r$.

Case 3 (t is a parallel node with left child ℓ and right child r). In this case, we look up π_ℓ and π_r as in Case 2, and we compute π_t such that $\sigma(\pi_t)$ dominates $\sigma(\pi_\ell) \boxplus \sigma(\pi_r)$ using the Merge Dominator Lemma (Lemma 6.27). Following Lemma 7.8(ii), we return π_t .

Finally, we return π_r , the topological order (of $G_r = G$) computed for r , the root of T . Observations 7.5 and 7.6 ensure that it is sufficient to compute in each of the above cases a set $\Pi_t^* \subseteq \Pi(G_t)$ with the following property. For each $\pi_t \in \Pi(G_t)$, there is a $\pi_t^* \in \Pi_t^*$ such that $\sigma(\pi_t^*) \prec \sigma(\pi_t)$. By Lemma 7.8, we know that we can always find such a set of size one which is precisely what we compute in each of the above cases. Correctness of the algorithm follows. Since T has $\mathcal{O}(n)$ nodes and each of the above cases can be handled in at most $\mathcal{O}(n)$ time by Lemma 6.27, we have that the total runtime of the algorithm is $\mathcal{O}(n^2)$. \square

Our algorithm in fact works for the more general problem of computing the *weighted* cutwidth of a series parallel digraph which we now define formally.

Definition 7.10. Let G be a directed acyclic graph and $\omega: A(G) \rightarrow \mathbb{N}$ be a weight function. For a topological order $\pi \in \Pi(G)$ of G , the *weighted cutwidth* of (π, ω) is defined as

$$\text{wcutw}(\pi, \omega) := \max_{i \in [n-1]} \sum_{\substack{vw \in A(G) \\ \pi(v) \leq i, \pi(w) > i}} \omega(vw),$$

and the *weighted cutwidth* of (G, ω) is $\text{wcutw}(G, \omega) := \min_{\pi \in \Pi(G)} \text{wcutw}(\pi, \omega)$.

The corresponding computational problem is defined as follows.

WEIGHTED CUTWIDTH OF SERIES PARALLEL DIGRAPHS

Input: A series parallel digraph G and an arc-weight function $\omega: A(G) \rightarrow \mathbb{N}$.

Question: What is the weighted cutwidth of (G, ω) ?

Corollary 7.11. Let G be an SPD on n vertices and $\omega: A(G) \rightarrow \mathbb{N}$ an arc-weight function. There is an algorithm that computes in time $\mathcal{O}(n^2)$ the weighted cutwidth of (G, ω) , and outputs a topological ordering that achieves the upper bound.

7.2 Modified Cutwidth of SPDs

We now show how to use the algorithm for computing the weighted cutwidth of series parallel digraphs from Corollary 7.11 to give an algorithm that computes the *modified cutwidth* of a series parallel digraph on n vertices in time $\mathcal{O}(n^2)$. Recall that given a topological order v_1, \dots, v_n of a directed acyclic graph G , its modified cutwidth is the maximum over all $i \in [n - 1]$ of the number of arcs that have their tail vertex in $\{v_1, \dots, v_{i-1}\}$ and their head vertex in $\{v_{i+1}, \dots, v_n\}$, and that the modified cutwidth of G is the minimum modified cutwidth over all its topological orders. We are dealing with the following computational problem.

MODIFIED CUTWIDTH OF SERIES PARALLEL DIGRAPHS

Input: A series parallel digraph G .

Question: What is the modified cutwidth of G ?

To solve this problem, we will provide a transformation which allows for applying the algorithm for the WEIGHTED CUTWIDTH OF SPD's problem to compute the modified cutwidth. We would like to remark that this transformation is similar to one provided in [35], however some modifications are necessary to ensure that the digraph resulting from the transformation is an SPD.

Theorem 7.12. *Let G be an SPD on n vertices. There is an algorithm that computes in time $\mathcal{O}(n^2)$ the modified cutwidth of G , and outputs a topological ordering of G that achieves the upper bound.*

Proof. We give a transformation that enables us to solve MODIFIED CUTWIDTH OF SPD's with help of an algorithm that solves WEIGHTED CUTWIDTH OF SPD's.

Let $(G, (s, t))$ be an SPD on n vertices and m arcs. Again, we assume that G has no parallel arcs; if so, we simply subdivide all but one of the parallel arcs. This does not change the (modified) cutwidth, and keeps a digraph series parallel. We construct another digraph G' and an arc-weight function $\omega: A(G') \rightarrow \mathbb{N}$ as follows. For each vertex $v \in V(G) \setminus \{s, t\}$, we add to G' two vertices v_{in} and v_{out} . We add s and t to G' and write s as s_{out} and t as t_{in} . We add the following arcs to G' . First, for each $v \in V(G)$, we add an arc (v_{in}, v_{out}) and we let $\omega((v_{in}, v_{out})) := m + 1$. Next, for each arc $(v, w) \in A(G)$, we add an arc (v_{out}, w_{in}) to G' and we let $\omega((v_{out}, w_{in})) := 1$. For an illustration see Figure 7.2.

We observe that the size of G' is linear in the size of G , and then prove that if G' is obtained from applying the above transformation to a series parallel digraph, then G' is itself an SPD.

Observation 7.12.1. *Let G and G' be as above. Then, $n' := |V(G')| \leq 2|V(G)|$ and $|A(G')| \leq |A(G)| + |V(G)|$.*

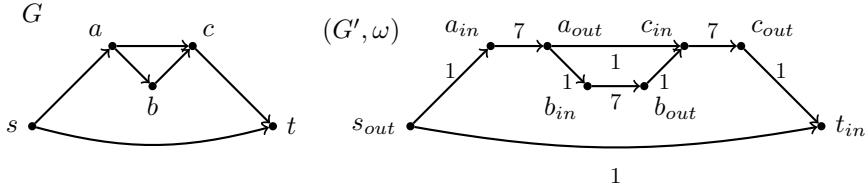


Figure 7.2: Illustration of the transformation given in the proof of Theorem 7.12. Note that in this case, \$m = 6\$, so the arcs between vertices \$v_{in}\$ and \$v_{out}\$ have weight 7.

Claim 7.12.2. *If \$G\$ is a series parallel digraph, then \$G'\$ as constructed above is an SPD.*

Proof. We prove the claim by induction on \$n\$, the number of vertices of \$G\$. For the base case when \$n = 2\$, we have that \$G\$ is a single arc in which case \$G'\$ is a single arc as well. Now suppose \$n > 2\$. Since \$n > 2\$, \$G\$ is obtained from two series parallel digraphs \$G_1\$ and \$G_2\$ via series or parallel composition. Since \$G\$ has no parallel arcs, we can use the induction hypothesis to conclude that the graphs \$G'_1\$ and \$G'_2\$ obtained via our construction are series parallel. Now, if \$G = G_1 \perp G_2\$, then it is immediate that \$G'\$ is series parallel. If \$G = G_1 \vdash G_2\$, then we have that in \$G'\$, the vertex that was constructed since \$t_1\$ and \$s_2\$ were identified, call this vertex \$x\$, got split into two vertices \$x_{in}\$ and \$x_{out}\$ with a directed arc of weight \$m + 1\$ pointing from \$x_{in}\$ to \$x_{out}\$. Call the series parallel digraph consisting only of this arc \$(X, (x_{in}, x_{out}))\$. We now have that \$G' = G'_1 \vdash X \vdash G'_2\$, so \$G'\$ is series parallel in this case as well. \$\square\$

We are now ready to prove the correctness of this transformation. To do so, we will assume that we are given an integer \$k\$ and we want to decide whether the modified cutwidth of \$G\$ is at most \$k\$.

Claim 7.12.3. *If \$G\$ has modified cutwidth at most \$k\$, then \$G'\$ has weighted cutwidth at most \$m + k + 1\$.*

Proof. Take a topological ordering \$\pi\$ of \$G\$ such that \$\text{mcutw}(\pi) \leq k\$. We obtain \$\pi'\$ from \$\pi\$ by replacing each vertex \$v \in V(G) \setminus \{s, t\}\$ by \$v_{in}\$ followed directly by \$v_{out}\$. Clearly, this is a topological order of \$G'\$. We show that the weighted cutwidth of this ordering is at most \$m + k + 1\$.

Let \$i \in [n' - 1]\$ and consider the cut between position \$i\$ and \$i + 1\$ in \$\pi'\$. We have to consider two cases. In the first case, there is some \$v \in V(G)\$ such that \$\pi'^{-1}(i) = v_{in}\$ and \$\pi'^{-1}(i + 1) = v_{out}\$. Then, there is an arc of weight \$m + 1\$ from \$v_{in}\$ to \$v_{out}\$ crossing this cut, and some other arcs of the form \$(u_{out}, w_{in})\$ for some arc \$(u, w) \in A(G)\$. All these arcs cross position \$\pi(v)\$ in \$\pi\$, so since \$\text{mcutw}(\pi) \leq k\$, there are at most \$k\$ of them. Furthermore, for each such arc we have that \$\omega((u_{out}, w_{in})) = 1\$ by construction, so the total weight of this cut is at most \$m + k + 1\$.

In the second case, we have that $\pi'^{-1}(i) = v_{out}$ and $\pi'^{-1}(i+1) = w_{in}$ for some $v, w \in V(G)$, $v \neq w$. By construction, we have that $\pi(w) = \pi(v) + 1$. Hence, any arc crossing the cut between i and $i+1$ in π' is of one of the following forms.

- (i) It is (x_{out}, y_{in}) for some $(x, y) \in A(G)$ with $\pi(x) < \pi(v)$ and $\pi(y) > \pi(v)$, or
- (ii) it is (x_{out}, y_{in}) for some $(x, y) \in A(G)$ with $\pi(x) < \pi(w)$ and $\pi(y) > \pi(w)$, or
- (iii) it is (v_{out}, w_{in}) .

Since $\text{mcutw}(G) \leq k$, there are at most k arcs of the first and second type, and since G has no parallel arcs, there is at most one arc of the third type. By construction, all these arcs have weight one, so the total weight of this cut is $2k + 1 \leq m + k + 1$.

Claim 7.12.4. *If G' has weighted cutwidth at most $m + k + 1$, then G has modified cutwidth at most k .*

Proof. Let π' be a topological order of G' such that $\text{wcutw}(\pi', \omega) \leq m + k + 1$. First, we claim that for all $v \in V(G) \setminus \{s, t\}$, we have that $\pi'(v_{out}) = \pi'(v_{in}) + 1$. Suppose not, for some vertex v . If we have that $\pi'(v_{in}) < \pi'(w_{in}) < \pi'(v_{out})$ for some $w \in V(G) \setminus \{s, t\}$ and $w \neq v$, then the cut between $\pi'(w_{in})$ and $\pi'(w_{in}) + 1$ has weight at least $2m + 2$: the two arcs (v_{in}, v_{out}) and (w_{in}, w_{out}) cross this cut, and they are of weight $m + 1$ each. Similarly, if $\pi'(v_{in}) < \pi'(w_{out}) < \pi'(v_{out})$, then the cut between $\pi'(w_{out}) - 1$ and $\pi'(w_{out})$ has weight at least $2m + 2$. Since $2m + 2 > m + k + 1$, we have a contradiction in both cases.

We define a linear ordering π of G as follows. We let $\pi(s) := 1$, $\pi(t) := n$, and for all $v, w \in V(G) \setminus \{s, t\}$, we have $\pi(v) < \pi(w)$ if and only if $\pi'(v_{in}) < \pi'(w_{in})$. It is clear that π is a topological ordering of G ; we show that π has modified cutwidth at most k . Consider an arc (x, y) that crosses a vertex v in π , i.e. we have that $\pi(x) < \pi(v) < \pi(y)$. We have just argued that $\pi'(v_{out}) = \pi'(v_{in}) + 1$, so we have that the arc (x_{out}, y_{in}) crosses the cut between v_{in} and v_{out} in π' . Recall that there is an arc of weight $m + 1$ from v_{in} to v_{out} , so since $\text{wcutw}(\pi', \omega) \leq m + k + 1$, we can conclude that in π , there are at most $(m + k + 1) - (m - 1) = k$ arcs crossing the vertex v in π . \square

Now, to compute the modified cutwidth of G , we run the above described transformation to obtain (G', ω) , and compute a topological order that gives the smallest weighted cutwidth of (G', ω) using Corollary 7.11. We can then follow the argument given in the proof of Claim 7.12.4 to obtain a topological order for G that gives the smalles modified cutwidth of G .

By Claim 7.12.2, G' is an SPD, so we can indeed apply the algorithm of Corollary 7.11 to solve the instance (G', ω) . Correctness follows from Claims 7.12.3 and 7.12.4. By Observation 7.12.1, $|V(G')| = \mathcal{O}(|V(G)|) = \mathcal{O}(n)$, and clearly, (G', ω) can be constructed in time $\mathcal{O}(|V(G)| + |A(G)|)$; so the overall runtime of this procedure is at most $\mathcal{O}(n^2)$. \square

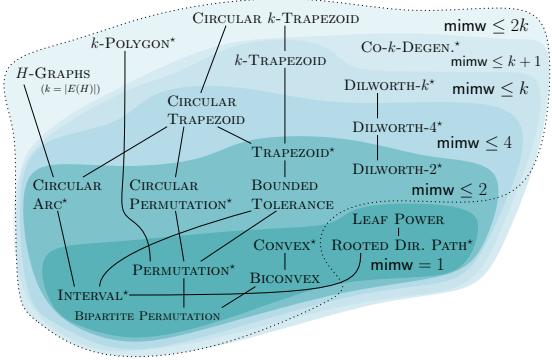
7.3 Further Applications of the MDL?

In this chapter we have seen some direct algorithmic applications of the Merge Dominator Lemma. Naturally, it would be interesting to see whether it can be used in the computation of other width measures, directed or undirected, in general or restricted to special classes of graphs. For instance, can the Merge Dominator Lemma be of help in designing XP algorithms for computing width measures parameterized by the treewidth of the input graph?

Lastly, we present one more open problem regarding the computation of width measures of series parallel digraphs. The *vertex separation number* of a topological order is the maximum over all cuts induced by the order of the number of vertices on the left side that have a neighbor on the right side. Finding a topological order that minimizes the vertex separation number corresponds to an important problem in compiler optimization, specifically to a problem related to register allocation: we are given a set of expressions (a “basic block” or “straight-line code”) that have certain dependencies among each other and the task is to find a sequence of executing these expressions, respecting the dependencies, such that the number of used registers is minimized. The dependencies among these expressions form an acyclic digraph and any allowed schedule is a topological ordering. This problem was shown to be NP-hard by Sethi [262] while Kessler [184] gave a $2^{\mathcal{O}(n)}$ time exact algorithm, improving over the $n^{\mathcal{O}(n)}$ naive brute-force approach. Sethi and Ullman [263] showed in 1970 that the problem is linear time solvable if the acyclic digraph is a tree which (to the best of our knowledge) is the only known polynomial-time case. It seems that with the help of the Merge Dominator Lemma, we might be able to obtain a polynomial-time algorithm for this problem on series parallel digraphs. However, the application cannot be as immediate as in the case of cutwidth and modified cutwidth. The vertex separation number counts *vertices* rather than edges (as it is the case for cutwidth and modified cutwidth), and in a parallel composition, the sources of two SPDs are being identified. In a straightforward approach, this results in overcounting the contribution of the source vertex to several cuts, which is the main obstacle that needs to be overcome.

Part III

Computing With Width Measures



Non-Local Problems on Mim-Width

Until recently, the only problems known to be XP-time solvable parameterized by mim-width¹ were *locally checkable*. Recall that by locally checkable we mean that their solutions can be verified by inspecting their interactions with the direct neighborhood of each vertex separately; for details see Section 3.6. It is therefore natural to ask whether similar results can be shown for problems concerning graph properties that are *not* locally checkable, which are the topic of this chapter.

Such problems typically impose a *global* structure on the solutions, such as a connectivity requirement. Arguably the simplest connected graphs are *paths*, and in Section 8.2, we study several problems that are related to finding *induced* paths in graphs, namely LONGEST INDUCED PATH, INDUCED DISJOINT PATHS, and *H*-INDUCED TOPOLOGICAL MINOR. Such algorithms are unlikely to exist for the ‘non-induced’ counterparts: hardness for HAMILTONIAN CYCLE (and therefore² LONGEST PATH) on graphs of linear mim-width 1 is shown in the next chapter, and DISJOINT PATHS is NP-complete on interval graphs [229], which are also of linear mim-width 1. Another commonly formulated global constraint is that of *acyclicity*. In Section 8.3 we give an XP-algorithm for FEEDBACK VERTEX SET parameterized by mim-width.

8.1 Notation and Minimal Vertex Covers

We introduce some concepts and notation that are used throughout the remainder of this chapter, and then prove one of the elementary tools that we need in all algorithms presented in this section, namely the *Minimal Vertex Covers Lemma*.

¹Note that in the more informal parts of this text, we simply say ‘parameterized by mim-width’ rather than ‘parameterized by the mim-width of a given branch decomposition of the input graph’ which strictly speaking is the correct term.

²Note that there is a standard reduction to HAMILTONIAN PATH that increases the linear mim-width by at most 1.

Definition 8.1 (Boundary). Let G be a graph and $A, B \subseteq V(G)$ such that $A \cap B = \emptyset$. We let $\text{bd}_B(A)$ be the set of vertices in A that have a neighbor in B , i.e. $\text{bd}_B(A) := \{v \in A \mid N(v) \cap B \neq \emptyset\}$. We define $\text{bd}(A) := \text{bd}_{V(G) \setminus A}(A)$ and call $\text{bd}(A)$ the *boundary* of A in G .

Definition 8.2 (Crossing Graph). Let G be a graph and $A, B \subseteq V(G)$. If $A \cap B = \emptyset$, we define the graph $G_{A,B} := G[\text{bd}_B(A), \text{bd}_A(B)]$ to be the *crossing graph from A to B* .

If (T, \mathcal{L}) is a branch decomposition of G and $t_1, t_2 \in V(T)$ such that the crossing graph $G_{V_{t_1}, V_{t_2}}$ is defined, we use the shorthand $G_{t_1, t_2} := G_{V_{t_1}, V_{t_2}}$. We use the analogous shorthand notations $G_{t_1, \overline{t_2}} := G_{V_{t_1}, \overline{V_{t_2}}}$ and $G_{\overline{t_1}, t_2} := G_{\overline{V_{t_1}}, V_{t_2}}$ (whenever these graphs are defined). For the frequently arising case when we consider $G_{t, \overline{t}}$ for some $t \in V(T)$, we refer to this graph as the *crossing graph w.r.t. t* .

Let G be a graph. We now prove that given a set $A \subseteq V(G)$, the number of minimal vertex covers in $G[A, V(G) \setminus A]$ is bounded by $n^{\text{mim}(A)}$. This observation is crucial to argue that we only need to store $n^{\mathcal{O}(w)}$ entries at each node in the branch decomposition in all algorithms we design, where w is the mim-width of the given branch decomposition.

Notice that the bound on the number can be easily obtained by combining two results, [17, Lemma 1] and [281, Theorem 3.5.5]; however, an enumeration algorithm is not given explicitly. To be self-contained, we state and prove it here.

Corollary 8.3 (Minimal Vertex Covers Lemma). *Let H be a bipartite graph on n vertices with a bipartition (A, B) . The number of minimal vertex covers of H is at most $n^{\text{mim}(A)}$, and the set of all minimal vertex covers of H can be enumerated in time $n^{\mathcal{O}(\text{mim}(A))}$.*

Proof. Let $w := \text{mim}(A)$. For each vertex set $R \subseteq A$ with $|R| \leq w$, let $X_R \subseteq A$ be the set of all vertices having a neighbor in $B \setminus N(R)$. We enumerate the sets in

$$\mathcal{M} = \{N(R) \cup X_R : R \subseteq A, |R| \leq w\}.$$

Clearly, we can enumerate them in time $n^{\mathcal{O}(w)}$. It is not difficult to see that each set in \mathcal{M} is a minimal vertex cover. We claim that \mathcal{M} is the set of all minimal vertex covers in H .

We use the result by Belmonte and Vatshelle [17, Lemma 1] that for a graph G and $A \subseteq V(G)$, $\text{mim}(A) \leq k$ if and only if for every $S \subseteq A$, there exists $R \subseteq S$ such that $N(R) \cap (V(G) \setminus A) = N(S) \cap (V(G) \setminus A)$ and $|R| \leq k$.

Let U be a minimal vertex cover of H . Clearly, every vertex in $A \setminus U$ has no neighbors in $B \setminus U$, as U is a vertex cover. Therefore, by the result of Belmonte and Vatshelle, there exists $R \subseteq A \setminus U$ such that $|R| \leq w$ and $N(R) \cap B = N(A \setminus U) \cap B = U \cap B$. Clearly, $U \cap A = X_R$; if a vertex in $U \cap A$ has no neighbors in $B \setminus U$, then we can remove it from the vertex cover. Therefore, $U \in \mathcal{M}$, as required. \square

8.2 Induced Path Problems on Mim-Width

In this section, we study problems related to finding *induced* paths in graphs, namely LONGEST INDUCED PATH, INDUCED DISJOINT PATHS and H -INDUCED TOPOLOGICAL MINOR, all parameterized by mim-width. Although their ‘non-induced’ counterparts are more deeply studied in the literature, these induced variants have received considerable attention as well. Before we discuss our results and methods of this chapter, we briefly survey results for exact algorithms to these three problems on graph classes studied so far.

For the first problem, Gavril [139] showed that LONGEST INDUCED PATH can be solved in polynomial time for graphs without induced cycles of length at least q for fixed q (the running time was improved by Ishizeki et al. [165]), while Kratsch et al. [197] gave a polynomial-time algorithm on AT-free graphs. Kang et al. [176] recently showed that those classes have unbounded mim-width. On the other hand, graphs of bounded mim-width are not necessarily graphs without induced cycles of length at least q or AT-free graphs. The second problem is derived from the well-known DISJOINT PATHS problem, asking for k pairwise disjoint paths between k pairs of terminal vertices. For fixed k , DISJOINT PATHS was famously shown to be solvable in $\mathcal{O}(n^3)$ time by Robertson and Seymour [256], which was improved to $\mathcal{O}(n^2)$ by Kawabaraayashi et al. [182]; while if k is part of the input then it is NP-complete on graphs of linear mim-width 1 (interval graphs) [229]. In contrast, on general graphs, INDUCED DISJOINT PATHS is NP-complete already for $k = 2$ paths [181]. In this chapter we consider the number of paths k as part of the input. In this setting INDUCED DISJOINT PATHS is NP-complete on claw-free graphs, as shown by Fiala et al. [123], while Golovach et al. [145] gave a linear-time algorithm for circular-arc graphs. For the third problem, H -INDUCED TOPOLOGICAL MINOR, we consider H to be a fixed graph. This problem, and also INDUCED DISJOINT PATHS, were both shown solvable in polynomial time on chordal graphs by Belmonte et al. [15], and on AT-free graphs by Golovach et al. [144].

We show that LONGEST INDUCED PATH, INDUCED DISJOINT PATHS and H -INDUCED TOPOLOGICAL MINOR for fixed H can be solved in time $n^{\mathcal{O}(w)}$ on n -vertex graphs, given a branch decomposition of mim-width w .

What makes our algorithms work is an analysis of the structure induced by a solution to the problem on a cut in the branch decomposition. There are two ingredients. First, in all the problems we investigate, we are able to show the following. Let (A, B) be a cut induced by an edge of a given branch decomposition of mim-width w of an input graph. It is sufficient to consider induced subgraphs on at most $\mathcal{O}(w)$ vertices as intersections of solutions with the edges crossing (A, B) . For instance, in the LONGEST INDUCED PATHS problem, an induced path is a target solution, and Figure 8.1 describes a situation where an induced path crosses a cut. We argue that an induced path cannot cross a cut many times if there is no large induced matching between vertex sets A and B of the cut (A, B) . Such an intersection is always an

induced disjoint union of paths. Thus, we enumerate all subgraphs of size at most $\mathcal{O}(w)$, which are induced disjoint unions of paths, and these will be used as indices of our table.

However, a difficulty arises if we recursively ask for a given cut and such an intersection of size at most $\mathcal{O}(w)$, whether there is an induced disjoint union of paths of certain size in the union of one part and edges crossing the cut, whose intersection on the crossing edges is the given subgraph. The reason is that there are an unbounded number of vertices in each part of the cut that are not contained in the given subgraph of size $\mathcal{O}(w)$ but still have neighbors in the other part. We need to control these vertices in such a way that they do not further create an edge in the solution. This is achieved using vertex covers of the bipartite graph induced by edges crossing the cut. Roughly speaking, if there is a valid partial solution, then there is a vertex cover of such a bipartite graph, which covers all other edges not contained in the given subgraph. The point is that there are only $n^{\mathcal{O}(w)}$ many minimal vertex covers of such a bipartite graph with maximum induced matching size w , by the Minimal Vertex Covers Lemma (Corollary 8.3). Based on these two results, each table will consist of a subgraph of size $\mathcal{O}(w)$ and a vertex cover of the remaining part of the bipartite graph, and we check whether there is a (valid) partial solution to the problem with respect to given information. We can argue that we need to store at most $n^{\mathcal{O}(w)}$ table entries in the resulting dynamic programming scheme and that each of them can be computed in time $n^{\mathcal{O}(w)}$ as well. This technique will also be crucially employed in the result of Chapter 8.3, in the algorithm that solves FEEDBACK VERTEX SET in $n^{\mathcal{O}(w)}$ time in the same parameterization.

The strategy for INDUCED DISJOINT PATHS is very similar to the one for LONGEST INDUCED PATH. The only thing to additionally consider is that in the disjoint union of paths, which is guessed as the intersection of a partial solution and edges crossing a cut, we need to remember which path is a subpath of the path connecting a given pair. We lastly provide a one-to-many reduction from H -INDUCED TOPOLOGICAL MINOR to INDUCED DISJOINT PATHS, that runs in polynomial time, and show that it can be solved in time $n^{\mathcal{O}(w)}$. Similar reductions have been shown earlier (see e.g. [15, 144]) but we include it here for completeness.

8.2.1 Longest Induced Path

For a disjoint union of paths P , we refer to its *size* as the number of its vertices, i.e. $|P| := |V(P)|$. If P has only one component, we use the terms ‘size’ and ‘length’ interchangeably. We now give an $n^{\mathcal{O}(w)}$ time algorithm for the following parameterized problem.

LONGEST INDUCED PATH (LIP) parameterized by $w := \text{mimw}(T, \mathcal{L})$

<i>Input:</i>	A graph G with branch decomposition (T, \mathcal{L}) and an integer k
<i>Question:</i>	Does G contain an induced path of length at least k ?

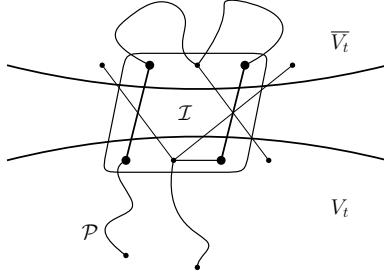


Figure 8.1: The intersection of an induced path \mathcal{P} with $G[V_t \cup \text{bd}(\bar{V}_t)]$, which is an induced disjoint union of paths \mathcal{I} . The subgraph S to be used as an index for the corresponding table entry consists of the boldface vertices and edges in \mathcal{I} .

Before we describe the algorithm, we observe the following. Let G be a graph and $A \subseteq V(G)$ with $\text{mim}(A) = w$ and let P be an induced path in G . Then the subgraph induced by edges of P in $G_{A, V(G) \setminus A}$ and vertices incident with these edges has size linearly bounded by w . The following lemma provides a bound of this size.

Lemma 8.4. *Let p be a positive integer and let F be a disjoint union of paths such that each component of F contains an edge. If $|V(F)| \geq 4p$, then F contains an induced matching of size at least p .*

Proof. We prove the lemma by induction on p . If $p = 1$, then it is clear. We may assume $p \geq 2$. Suppose F contains a connected component C with at most 4 vertices. Then $F - V(C)$ contains at least $4(p-1)$ vertices, and thus it contains an induced matching of size at least $p-1$ by the induction hypothesis. As C contains an edge, F contains an induced matching of size at least p . Thus, we may assume that each component of F contains at least 5 vertices. Let us choose a leaf v of F , and let v_1 be the neighbor of v , and v_2 be the neighbor of v_1 other than v . Since each component of $F - \{v, v_1, v_2\}$ contains at least one edge, we can apply induction to conclude that $F - \{v, v_1, v_2\}$ contains an induced matching of size at least $p-1$. Together with vv_1 , F contains an induced matching of size at least p . \square

Remark 8.5. Unless stated otherwise, any path P (or equivalently, a component of a disjoint union of paths) we refer to throughout the remainder of this section is considered to be nontrivial, i.e. P contains at least one edge.

Before we give the description of the dynamic programming algorithm, we first observe how a solution \mathcal{P} , i.e. an induced path in G , interacts with the graph $G[V_t \cup \text{bd}(\bar{V}_t)]$, for some $t \in V(T)$. The intersection of \mathcal{P} with $G[V_t \cup \text{bd}(\bar{V}_t)]$ is an induced disjoint union of paths which we will denote by \mathcal{I} in the following. To keep the number of possible table entries bounded by $n^{\mathcal{O}(w)}$, we have to focus on the interaction of \mathcal{I} with the crossing graph $G_{t, \bar{t}}$ w.r.t. t , in particular the intersection of \mathcal{I} with its

edges. Note that after removing isolated vertices, \mathcal{I} induces a disjoint union of paths on $G_{t,\bar{t}}$ which throughout the following we will denote by S . For an illustration see Figure 8.1.

Throughout the following, we call vertices of \mathcal{I} that are *not* contained in $V(S)$ *intermediate vertices* w.r.t. the cut (V_t, \bar{V}_t) . If the cut is clear from the context, we simply call them intermediate vertices. Note that by definition there cannot be any edges between intermediate vertices on opposite sides of the boundary. This property of \mathcal{I} can be captured by considering a minimal vertex cover M of the bipartite graph $G_{t,\bar{t}} - V(S)$. We remark that the vertices in M play different roles, depending on whether they lie in $M \cap V_t$ or $M \cap \bar{V}_t$. We therefore define the following two sets.

- $M_t^{\text{in}} := M \cap V_t$ is the set of vertices that must be avoided by \mathcal{I} .
- $M_t^{\text{out}} := M \cap \bar{V}_t$ is the set of vertices that must be avoided by a partial solution (e.g. the intersection of \mathcal{P} with $G[\bar{V}_t]$) to be combined with \mathcal{I} to ensure that their combination does not use any edges in $G_{t,\bar{t}} - V(S)$.

Furthermore, \mathcal{I} also indicates how the vertices in $S[V_t]$ that have degree one in S are joined together in the graph G_t (possibly outside $\text{bd}(V_t)$). This gives rise to a collection of vertex pairs Q , which we will refer to as *pairings*, with the interpretation that $(s, t) \in Q$ if and only if there is a path from s to t in $\mathcal{I}[V_t]$.

The description given above immediately tells us how to index the table entries in the dynamic programming table tab to keep track of all possible partial solutions in the graph $G[V_t \cup \text{bd}(\bar{V}_t)]$: We set the table entry $\text{tab}[t, (S, M, Q), i, j] = 1$, where $i \in [0..n]$ and $j \in [0..2]$, if and only if the following conditions are satisfied. For an illustration of the table indices, see Figure 8.2.

- (i) There is a set of induced paths \mathcal{I} of total size i in $G[V_t \cup \text{bd}(\bar{V}_t)]$ such that \mathcal{I} has j degree one endpoints in G_t .
- (ii) $E(\mathcal{I}) \cap E(G_{t,\bar{t}}) = E(S)$.
- (iii) M is a minimal vertex cover of $G_{t,\bar{t}} - V(S)$ such that $V(\mathcal{I}) \cap M = \emptyset$. (Recall that $M = M_t^{\text{in}} \cup M_t^{\text{out}}$.)
- (iv) Let D denote the set of the vertices in $S[V_t]$ that have degree one in S . Let $Q = \{(s_1, t_1), \dots, (s_\ell, t_\ell)\}$ be a partition of all but j vertices of D into pairs, throughout the following called a *pairing*, such that if we contract all edges in $\mathcal{I} - E(G_{t,\bar{t}})$ from \mathcal{I} incident with at least one vertex not in S (we denote the resulting graph as $S \odot Q$) we obtain the same graph as when adding $\{s_k, t_k\}$ to S , for each $k \in [\ell]$.

Regarding (iv), observe that $|D| = 2\ell + j$ and that there are j unpaired vertices in Q , each of which is connected to a degree one endpoint of \mathcal{I} in G_t . For notational

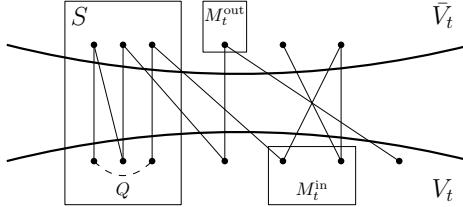


Figure 8.2: A crossing graph $G_{t,\bar{t}}$ and the structures associated with the table indices of the algorithm for LONGEST INDUCED PATH. Note that by (i) and (iv) it follows that if the table entry corresponding to the above structures is 1, then $j = 0$: Since both degree one endpoints in $S[V_t]$ are paired, this means that the corresponding set of induced paths \mathcal{I} has no degree one endpoints in $G[V_t]$.

convenience, we will denote by tab_t all table entries that have the node $t \in V(T)$ as the first index.

We now show that the solution to LONGEST INDUCED PATH can be obtained from a table entry corresponding to the root r of T and hence ensure that the information stored in tab is sufficient.

Proposition 8.6. *G contains an induced path of length i if and only if*

$$\text{tab}[r, (\emptyset, \emptyset, \emptyset), i, 2] = 1.$$

Proof. Suppose G contains an induced path \mathcal{P} of length i and consider the root r of (T, \mathcal{L}) . Clearly, $G[V_r \cup \text{bd}(\bar{V}_r)] = G$, since $V_r = V(G)$ and $\text{bd}(\bar{V}_r) = \emptyset$. Together with the fact that \mathcal{P} is a path (and hence has two degree one endpoints in $G_r = G$), it follows that \mathcal{P} satisfies Condition (i) for a table entry to be set to 1. Since $\text{bd}(\bar{V}_r) = \emptyset$, it follows that for any index in tab_r , $S = \emptyset$, $M = \emptyset$ and $Q = \emptyset$. We can conclude that $\text{tab}[r, (\emptyset, \emptyset, \emptyset), i, 2] = 1$.

Now suppose $\text{tab}[r, (\emptyset, \emptyset, \emptyset), i, 2] = 1$. Then there is a set of induced paths \mathcal{I} of total size i in G by (i) having two degree one endpoints in $G_r = G$. The latter allows us to conclude that \mathcal{I} is in fact a single path. \square

Throughout the following, we denote by \mathcal{S}_t the set of all sets of induced disjoint paths in $G_{t,\bar{t}}$ on at most $4w$ vertices without isolated vertices (which includes all possible intersections of partial solutions with $G_{t,\bar{t}}$ by Lemma 8.4), and for $S \in \mathcal{S}_t$ denote by $\mathcal{M}_{t,S}$ the set of all minimal vertex covers of $G_{t,\bar{t}} - V(S)$ and by $\mathcal{Q}_{t,S}$ the set of all pairings of degree one vertices in $S[\text{bd}(V_t)]$. We now argue that the number of such entries is bounded by a polynomial in n whose degree is $\mathcal{O}(w)$.

Proposition 8.7. *For each $t \in V(T)$, there are at most $n^{\mathcal{O}(w)}$ table entries in tab_t and they can be enumerated in time $n^{\mathcal{O}(w)}$.*

Proof. Note that each index is an element of $\mathcal{S}_t \times \mathcal{M}_{t,S} \times \mathcal{Q}_{t,S} \times [0..n] \times [0..2]$. Since the size of each maximum induced matching in $G_{t,\bar{t}}$ is at most w , we know by Lemma

8.4 that the size of each index S is bounded by $4w$, so $|\mathcal{S}_t| \leq \mathcal{O}(n^{4w})$. By the Minimal Vertex Covers Lemma (Corollary 8.3), $|\mathcal{M}_{t,S}| \leq n^{\mathcal{O}(w)}$. Since the number of vertices in S is bounded by $4w$, we know that $|\mathcal{Q}_{t,S}| \leq w^{\mathcal{O}(w)}$ and since $i \in [0..n]$ and $j \in [0..2]$, we can conclude that the number of table entries for each $t \in V(T)$ is at most

$$\mathcal{O}(n^{4w}) \cdot n^{\mathcal{O}(w)} \cdot w^{\mathcal{O}(w)} \cdot (n+1) \cdot 3 = n^{\mathcal{O}(w)},$$

as claimed. Clearly, all elements in \mathcal{S}_t and $\mathcal{Q}_{t,S}$ can be enumerated in time $n^{\mathcal{O}(w)}$ and by the Minimal Vertex Covers Lemma, we know that all elements in $\mathcal{M}_{t,S}$ can be enumerated in time $n^{\mathcal{O}(w)}$ as well. The claimed time bound on the enumeration of the table indices follows. \square

In the remainder of the proof we will describe how to fill the table entries from the leaves of T to its root, asserting the correctness of the updates in the table. Together with Proposition 8.6, this will yield the correctness of the algorithm.

Leaves of T . Let $t \in V(T)$ be a leaf node of T and let $v = \mathcal{L}^{-1}(t)$. We observe that any nonempty intersection of an induced disjoint union of paths in $G_{t,\bar{t}}$ is a single edge (whose length/size is two) using v or a path on two edges (whose length/size is three) whose middle vertex is v . Hence, all indices in tab_t that are set to 1 (with nonempty S) have the following properties: Either S is a single edge using v , and we denote the set of all such edges by $\mathcal{S}_{t,v}^1$ or a path on two edges with v as the middle vertex, and we denote the corresponding set of such paths as $\mathcal{S}_{t,v}^2$. For each $S \in \mathcal{S}_{t,v}^1 \cup \mathcal{S}_{t,v}^2$, the corresponding minimal vertex cover of $G_{t,\bar{t}} - V(S)$ is empty, since $v \in V(S)$ and no edges remain in $G_{t,\bar{t}}$ when we remove v . Since v is the only vertex in G_t , $Q = \emptyset$ in both of these cases. If $S \in \mathcal{S}_{t,v}^1$, then $j = 1$ and if $S \in \mathcal{S}_{t,v}^2$, then $j = 0$. Additionally, a table entry is set to 1 if $S = \emptyset$ and $i = 0$ and since the solution is empty in this case, $Q = \emptyset$ and $j = 0$. The two corresponding minimal vertex covers are $\{v\}$ and $N(v)$. Hence, we set the table entries in the leaves as follows.

$$\text{tab}[t, (S, M, Q), i, j] := \begin{cases} 1, & \text{if } S \in \mathcal{S}_{t,v}^1, M = \emptyset, Q = \emptyset, i = 2 \text{ and } j = 1 \\ 1, & \text{if } S \in \mathcal{S}_{t,v}^2, M = \emptyset, Q = \emptyset, i = 3 \text{ and } j = 0 \\ 1, & \text{if } S = \emptyset, M \in \{\{v\}, N(v)\}, Q = \emptyset, i = 0 \text{ and } j = 0 \\ 0, & \text{otherwise} \end{cases}$$

Internal nodes of T . Let $t \in V(T)$ be an internal node of T , let $(S, M, Q) \in \mathcal{S}_t \times \mathcal{M}_{t,S} \times \mathcal{Q}_{t,S}$, let $i \in [0..n]$ and $j \in [0..2]$. We show how to compute the table entry $\text{tab}[t, (S, M, Q), i, j]$ from table entries corresponding to the children a and b of t in T . To do so, we have to take into account the ways in which partial solutions for $G[V_a \cup \text{bd}(\bar{V}_a)]$ and $G[V_b \cup \text{bd}(\bar{V}_b)]$ interact. We therefore try all pairs of indices $\mathfrak{I}_a = ((S_a, M_a, Q_a), i_a, j_a)$, $\mathfrak{I}_b = ((S_b, M_b, Q_b), i_b, j_b)$ and for each such pair, first check whether it is ‘compatible’ with \mathfrak{I}_t : We say that \mathfrak{I}_a and \mathfrak{I}_b are *compatible with* \mathfrak{I}_t if and only if any partial solution \mathcal{I}_a represented by \mathfrak{I}_a for $G[V_a \cup \text{bd}(\bar{V}_a)]$ and

\mathcal{I}_b represented by \mathfrak{I}_b for $G[V_b \cup \text{bd}(\overline{V}_b)]$ can be combined to a partial solution \mathcal{I}_t for $G[V_t \cup \text{bd}(\overline{V}_t)]$ that is represented by the index \mathfrak{I}_t . We then set $\text{tab}_t[\mathfrak{I}_t] := 1$ if and only if we can find a compatible pair of indices $\mathfrak{I}_a, \mathfrak{I}_b$ as above such that $\text{tab}_a[\mathfrak{I}_a] = 1$ and $\text{tab}_b[\mathfrak{I}_b] = 1$.

Step 0 (Valid Index). We first check whether the index \mathfrak{I}_t can represent a valid partial solution of $G[V_t \cup \text{bd}(\overline{V}_t)]$. The definition of the table entries requires that $S \odot Q$ is a disjoint union of paths, so if $S \odot Q$ is not a disjoint union of paths, we set $\text{tab}_t[\mathfrak{I}_t] := 0$ and skip the remaining steps. In general, the number of degree one vertices in $V(S \odot Q) \cap V_t$ has to be equal to j and we can proceed as described in Steps 1-4, except for the following special cases.

Special Case 1 ($j = 2$, $V(S \odot Q) \cap V_t$ has 0 vertices of deg. 1 in $S \odot Q$).

This is the case when \mathcal{I} does not contain any edge in $E(G_{t,\bar{t}})$. It immediately follows that S has to be empty (and hence Q has to be empty). If not, we set $\text{tab}_t[\mathfrak{I}_t] = 0$ and skip the remaining steps. We would like to remark that the case when $j = 2$ and $S = \emptyset$ will have to be dealt with separately in Step 2, since $S \odot Q$ is the empty graph.

Special Case 2 ($j = 2$, $S \odot Q$ has a component C that is a path with 2 endpoints in V_t).

In this case, $S \odot Q$ has to consist of a single component (and \mathcal{I} of a single path): If there was another component C' in $S \odot Q$, C and C' could never be joined together to become a single path in the vertices of \overline{V}_t and hence we can never obtain a valid solution from the partial solution represented by this index. So if $S \odot Q$ has more than one component, we set $\text{tab}_t[\mathfrak{I}_t] = 0$ and skip the remaining steps, otherwise we do not have to pay any further attention to this case in the following computations.

Special Case 3 ($j = 0$ and $S = \emptyset$).

This is the case when \mathcal{I} does not use any vertex of V_t . We then set $\text{tab}_t[\mathfrak{I}_t] = 1$ if and only if $i = 0$ and skip the remaining steps.

Step 1 (Induced disjoint unions of paths). We now check whether S_a and S_b are compatible with S . We have to ensure that

- $S \cap G_{a,\bar{t}} = S_a \cap G_{a,\bar{t}}$,
- $S \cap G_{b,\bar{t}} = S_b \cap G_{b,\bar{t}}$ and
- $S_a \cap G_{a,b} = S_b \cap G_{a,b}$.

If these conditions are not satisfied, we skip the current pair of indices $\mathfrak{I}_a, \mathfrak{I}_b$. In the following, we use the notation $R = S_a \cap G_{a,b}$ ($= S_b \cap G_{a,b}$).

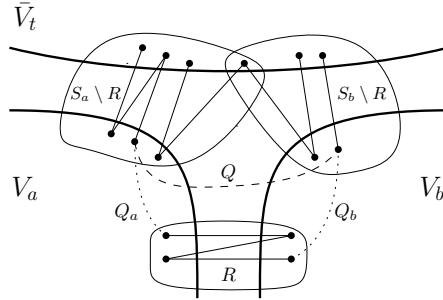


Figure 8.3: Step 2 of the join operation. Recall that $S_a \cap G_{a,b} = S_b \cap G_{a,b} = R$ by Step 1.

Step 2 (Pairings of degree one vertices in $S \odot Q$ and j). First, we deal with Special Case 1, i.e. $j = 2$ and $S = \emptyset$. We then check whether the graph obtained from taking R and adding an edge (and, if not already present, the corresponding vertices) for each pair in Q_a and Q_b is a single induced path. Note that we require the values of the integers j_a and j_b to be the number of endpoints of the resulting path in V_a and V_b , respectively.

Since the case $j = 0$ and $S = \emptyset$ is dealt with in Special Case 3 and S cannot be empty whenever $j = 1$, we may from now on assume that $S \neq \emptyset$ and hence $S \odot Q \neq \emptyset$.³

Consider the graph on vertex set $V(S) \cup V(R)$ whose edges consist of the edges in S and R together with the pairs in Q_a and Q_b . We then contract all edges in R and all edges that were added due to the pairings Q_a and Q_b and incident with a vertex not in S , and denote the resulting graph by \mathcal{H} . Then, Q_a and Q_b are compatible if and only if $\mathcal{H} = S \odot Q$. By the definition of the table entries (and since by Step 0, $S \odot Q$ is a disjoint union of paths) we can then see that Q_a, Q_b together with the edges of R connect the paired degree one vertices of Q as required. We furthermore need to ensure that the values of the integers j_a and j_b are the number of degree one endpoints in $\mathcal{H}[V_a]$ and $\mathcal{H}[V_b]$, respectively. For an illustration see Figure 8.3.

Step 3 (Minimal vertex covers). We now describe the checks we have to perform to ensure that M_a and M_b are compatible with M , which from now on we will denote by M_t to avoid confusion. For ease of exposition, we denote by \mathcal{I}_t , \mathcal{I}_a and \mathcal{I}_b (potential) partial solutions corresponding to \mathfrak{I}_t , \mathfrak{I}_a and \mathfrak{I}_b , respectively.

Recall that the purpose of the minimal vertex cover M_t is to ensure that no unwanted edges appear between vertices used by the partial solution \mathcal{I}_t and any partial solution of $G[\bar{V}_t \setminus \text{bd}(\bar{V}_t)]$ that can be combined with \mathcal{I}_t . Hence, when checking whether \mathcal{I}_a and \mathcal{I}_b can be combined to \mathcal{I}_t without explicitly having access to these

³Note that it could still happen that $Q = \emptyset$ but since this does not essentially influence the following argument, we assume that $Q \neq \emptyset$.

sets of induced disjoint paths, we have to make sure that the indices \mathcal{I}_a and \mathcal{I}_b assert the absence of unwanted edges — for any intersection of a partial solution with $G_{a,\bar{a}}$ and $G_{b,\bar{b}}$, as well as with $G_{a,b}$. Recall that $G_{a,\bar{a}} = G_{a,\bar{t}} \cup G_{a,b}$ and $G_{b,\bar{b}} = G_{b,\bar{t}} \cup G_{a,b}$.

We distinguish several cases, depending on where the unwanted edge might appear: First, between two intermediate vertices of partial solutions and second, between a vertex in S_a or S_b and an intermediate vertex. Step 3.1 handles the former and Step 3.2 the latter. In Step 3.1, we additionally have to distinguish whether the edge might appear in $G_{a,b}$ or in $G_{a,\bar{t}}$ (respectively, in $G_{b,\bar{t}}$).

In the following, we let $M_a^{\text{out}(b)} := M_a^{\text{out}} \cap V_b$ and $M_b^{\text{out}(a)} := M_b^{\text{out}} \cap V_a$.

Step 3.1.1 (intermediate-intermediate, $G_{a,b}$). We have to check that $M_a^{\text{out}(b)} \subseteq M_b^{\text{in}}$ and $M_b^{\text{out}(a)} \subseteq M_a^{\text{in}}$.

A vertex $v \in M_a^{\text{out}(b)}$ can have a neighbor $u \in V_a$ which is used as an intermediate vertex in \mathcal{I}_a . Hence, to avoid that the unwanted edge $\{u, v\}$ appears in the combined solution $\mathcal{I}_a \cup \mathcal{I}_b$, we have to make sure that v is not used by \mathcal{I}_b , which is asserted if $v \in M_b^{\text{in}}$. By a symmetric argument we justify that $M_b^{\text{out}(a)} \subseteq M_a^{\text{in}}$.

Step 3.1.2 (intermediate-intermediate, $G_{a,\bar{t}}$ or $G_{b,\bar{t}}$). We have to check the following two conditions, the first one regarding M_t^{in} and the second one regarding M_t^{out} .

- (a). $M_t^{\text{in}} \subseteq M_a^{\text{in}} \cup M_b^{\text{in}}$: By the definition of M_t^{in} , \mathcal{I}_t has to avoid the vertices in M_t^{in} . Hence, \mathcal{I}_a and \mathcal{I}_b have to avoid the vertices in M_t^{in} as well, which is ensured if for $v \in M_t^{\text{in}} \cap V_a$, we have that $v \in M_a^{\text{in}}$ and for $u \in M_t^{\text{in}} \cap V_b$, we have that $u \in M_b^{\text{in}}$.
- (b). For each vertex $v \in M_t^{\text{out}}$ having a neighbor x in V_a such that x is also contained in $V_a \setminus (V(S_a) \cup M_a^{\text{in}})$, we have that $v \in M_a^{\text{out}}$. Recall that by the definition of M_t^{out} , \mathcal{I}_t could use the vertex x as an intermediate vertex. If $x \notin V(S_a) \cup M_a^{\text{in}}$, this means that x might be used by \mathcal{I}_a as an intermediate vertex as well. Now, in a table entry representing a partial solution \mathcal{I}_a using x , this is signalized by having $v \in M_a^{\text{out}}$. We check the analogous condition for M_b^{out} .

Step 3.2 (intermediate-(S_a or S_b)). We require $V_a \cap N(V(S_b) \setminus V(S_a)) \subseteq M_a^{\text{in}}$ and $V_b \cap N(V(S_a) \setminus V(S_b)) \subseteq M_b^{\text{in}}$.

We justify the first condition and note that the second one can be argued for symmetrically. Clearly, \mathcal{I}_a cannot have a neighbor x of any vertex $v \in V(S_b)$ as an intermediate vertex, if \mathcal{I}_a is to be combined with \mathcal{I}_b . However, if $v \in V(S_a)$, then \mathcal{I}_a does not use x by Part ((ii)) of the definition of the table entries. Note that this includes all vertices in $V(R) \subseteq V(S_a)$. If on the other hand, $v \in V(S_b) \setminus V(S_a)$ then the neighbors of v have not been accounted for earlier, since v is not a vertex in the partial solution \mathcal{I}_a . Hence, we now have to assert that \mathcal{I}_a does not use x , the neighbor of v , and so we require that $x \in M_a^{\text{in}}$.

Step 4 (i). We consider all pairs of integers i_a, i_b such that $i = i_a + i_b - |V(R)|$. By Step 2, all vertices in R are used in the partial solution \mathcal{I}_t . They are counted twice, since they are both accounted for in \mathcal{I}_a and in \mathcal{I}_b .

Now, we let $\text{tab}_t[\mathfrak{J}_t] = 1$ if and only if there is a pair of indices $\mathfrak{J}_a, \mathfrak{J}_b$ passing all checks performed in Steps 1-4 above, such that $\text{tab}_a[\mathfrak{J}_a] = 1$ and $\text{tab}_b[\mathfrak{J}_b] = 1$. This finishes the description of the algorithm.

Proposition 8.8. *Let $t \in V(T)$. The table entries $\text{tab}_t[\mathfrak{J}_t]$ computed according to Steps 0-4 above are correct.*

Proof. Suppose there is partial solution \mathcal{I}_t , an induced disjoint union of paths in $G[V_t \cup \text{bd}(\overline{V}_t)]$. We claim that for any index \mathfrak{J}_t representing the partial solution \mathcal{I}_t , $\text{tab}_t[\mathfrak{J}_t] = 1$ after the join operation described by Steps 0-4, assuming as the induction hypothesis that the table values of the children a and b of t are computed correctly. Any index \mathfrak{J}_t representing \mathcal{I}_t has the following properties: $S = \mathcal{I}_t \cap G_{t,\bar{t}}$ and the pairing Q of the degree one vertices in $S[V_t]$ can be obtained by letting $(s, t) \in Q$ if and only if there is a path connecting s and t in $\mathcal{I}_t[V_t]$. We furthermore have that $i = |\mathcal{I}_t|$ and j is the number of degree one endpoints in $\mathcal{I}_t[V_t \setminus \text{bd}(V_t)]$. However, there might be several choices for the minimal vertex cover M of $G_{t,\bar{t}} - V(S)$. By the definition of the table entries, $M_t^{\text{in}} \subseteq V_t \setminus V(\mathcal{I}_t)$ and M_t^{out} contains all vertices of \overline{V}_t that have a neighbor in $V(\mathcal{I}_t) \setminus V(S)$. Throughout the remainder of the proof, we fix one such vertex cover M . Clearly, $\mathfrak{J}_t = ((S, M, Q), i, j)$ is a valid index according to the checks done in Step 0 and represents \mathcal{I}_t in the sense of the definition of the table entries.

We observe that \mathcal{I}_t induces partial solutions for the children a and b of t : We let $\mathcal{I}_a = \mathcal{I}_t \cap G[V_a \cup \text{bd}(\overline{V}_a)]$ and $\mathcal{I}_b = \mathcal{I}_t \cap G[V_b \cup \text{bd}(\overline{V}_b)]$. We now show how to obtain indices \mathfrak{J}_a and \mathfrak{J}_b representing \mathcal{I}_a and \mathcal{I}_b , respectively, that are compatible to be combined to the index \mathfrak{J}_t in the sense of Steps 1-4 of the algorithm presented above. In complete analogy to above, we obtain $S_a = \mathcal{I}_a \cap G_{a,\bar{a}}$, the pairing Q_a of degree one endpoints in $S_a[V_a]$, and the integers i_a and j_a and we proceed in the same way to obtain S_b, Q_b, i_b and j_b . Again, there will be several choices for the minimal vertex covers M_a of $G_{a,\bar{a}} - V(S_a)$ and M_b of $G_{b,\bar{b}} - V(S_b)$, of which we will choose one ‘representative’. For the details see further below. What we can immediately verify is that S_a and S_b are compatible with S in the sense of Step 1, that Q_a and Q_b are compatible with Q in the sense of Step 2, that the values of j_a and j_b are compatible with j , and that i_a and i_b are compatible with i in accordance with Step 4. Throughout the following, we denote by $R = S_a \cap S_b$ and note that R is contained in the crossing graph $G_{a,b}$.

We now explain how to construct a pair of minimal vertex covers M_a and M_b of $G_{a,\bar{a}} - V(S_a)$ and $G_{b,\bar{b}} - V(S_b)$, respectively, making sure that they are compatible with M in the sense of Step 3 of the algorithm description. Note that in the following, we only show how to construct M_a and we perform the symmetric steps to construct

M_b . Our strategy is as follows: We add vertices to M_a in consecutive stages and ensure in each stage that for each vertex x that we add to M_a ,

- x covers an edge that has not been covered by M_a so far, and
- each neighbor of x has another neighbor that is not contained in M_a .

Hence minimality of the resulting vertex cover will follow. We furthermore point out at each stage, which part of Step 3 in the description of the join operation it satisfies.

1. Let u be an intermediate vertex of \mathcal{I}_b and let x be a neighbor of u in V_a . Then, add x to M_a^{in} . Now, let u be an intermediate vertex of \mathcal{I}_a and x be a neighbor of \mathcal{I}_a in $\overline{V_a}$. Then, add x to M_a^{out} . In both cases, this covers the edge $\{u, x\}$. Since we do not add any more vertices to $M_a^{\text{out}(b)}$ in the remaining construction, the vertex u in the first case will never be added to M_a and since in the second case, u is an intermediate vertex, it will not be added to M_a either. Hence this stage of the construction cannot violate the minimality condition of M_a . Note that since we proceed symmetrically for M_b , the condition in Step 3.1.1 is also satisfied by this part of the construction.
2. We add any $x \in M_t^{\text{in}} \cap V_a$ to M_a^{in} . Since M_t is minimal, x covers an edge $\{u, x\}$ in $G_{a, \bar{t}}$, where $u \in \overline{V_t} \setminus V(S_t)$. Hence, the edge $\{u, x\}$ has not been covered so far by M_a . This ensures that the condition in Step 3.1.2 a) is met.
3. If a vertex x in $V_a \setminus V(S_a)$ has a neighbor u in $V(S_b) \setminus V(S_a)$, then add x to M_a^{in} , so x covers the edge $\{u, x\}$. Note that this vertex u is different from the one in Stage 2, as $u \in V(S_b) \setminus V(S_a)$ and hence u is either a vertex of S_t or it is *not* contained in $\overline{V_t}$. This asserts that the condition in Step 3.2 is satisfied.
4. Consider the subgraph G^* of $G_{a, \bar{a}} - V(S_a)$ on the edges that have not been covered by M_a so far. By Stage 1, this graph does not touch any vertex in \mathcal{I}_a or \mathcal{I}_b . We then add all vertices in $V(G^*) \cap V_a$ to M_a^{in} . This ensures that there is no vertex $v \in M_t^{\text{out}}$ that violates the condition of Step 3.1.2 b). Note that in this step, minimality of M_a is guaranteed as well, since all vertices in M_a^{out} have a neighbor in \mathcal{I}_a , and $M_a \cap V(\mathcal{I}_a) = \emptyset$.

By the above construction, in particular by Stage 4, we verify that M_a is a vertex cover of $G_{a, \bar{a}} - V(S_a)$ and in each stage, we checked that M_a remains minimal after adding the corresponding vertices.

We have shown how to derive from the partial solutions \mathcal{I}_a and \mathcal{I}_b a pair of table entries \mathfrak{I}_a , \mathfrak{I}_b that represent them. Since we assume inductively that the algorithm is correct for the children of t , we know that $\text{tab}_a[\mathfrak{I}_a] = 1$ and $\text{tab}_b[\mathfrak{I}_b] = 1$. By the description of the algorithm, this implies that $\text{tab}_t[\mathfrak{I}_t] = 1$ which concludes this direction of the proof.

For the other direction, suppose there exists an index \mathfrak{I}_t such that $\text{tab}_t[\mathfrak{I}_t] = 1$. By the description of the algorithm we can find pairs of indices \mathfrak{I}_a and \mathfrak{I}_b that are

compatible with \mathcal{I}_t and such that $\text{tab}_a[\mathcal{I}_a] = 1$ and $\text{tab}_b[\mathcal{I}_b] = 1$. Assuming for the induction hypothesis that the values of the children of t are computed correctly, we obtain the corresponding partial solutions \mathcal{I}_a and \mathcal{I}_b . Following the description of the algorithm, we can then see that \mathcal{I}_a and \mathcal{I}_b can be combined to a valid solution \mathcal{I}_t which is represented by \mathcal{I}_t . \square

By Propositions 8.6 and 8.8 and the fact that in the leaf nodes of T , we enumerate all possible partial solutions, we know that the algorithm we described is correct. Since by Proposition 8.7, there are at most $n^{\mathcal{O}(w)}$ table entries at each node of T (and they can be enumerated in time $n^{\mathcal{O}(w)}$), the value of each table entry in tab_t as above can be computed in time $n^{\mathcal{O}(w)} \cdot n^{\mathcal{O}(w)} \cdot n^{\mathcal{O}(1)} = n^{\mathcal{O}(w)}$, since each check described in Steps 0-4 can be done in time polynomial in n . Since additionally, $|V(T)| = \mathcal{O}(n)$, the total runtime of the algorithm is $n^{\mathcal{O}(w)} \cdot n^{\mathcal{O}(w)} \cdot \mathcal{O}(n) = n^{\mathcal{O}(w)}$ and we have the following theorem.

Theorem 8.9. *There is an algorithm that given a graph G on n vertices and a branch decomposition (T, \mathcal{L}) of G , solves LONGEST INDUCED PATH in time $n^{\mathcal{O}(w)}$, where w denotes the mim-width of (T, \mathcal{L}) .*

8.2.2 Induced Disjoint Paths

In this section, we build upon the ideas of the algorithm for LONGEST INDUCED PATH presented above to obtain an $n^{\mathcal{O}(w)}$ -time algorithm for the following parameterized problem.

INDUCED DISJOINT PATHS (IDP) parameterized by $w := \text{mimw}(T, \mathcal{L})$

- | | |
|------------------|--|
| <i>Input:</i> | A graph G with branch decomposition (T, \mathcal{L}) and pairs of vertices $(x_1, y_1), \dots, (x_k, y_k)$ of G . |
| <i>Question:</i> | Does G contain a set of vertex-disjoint induced paths P_1, \dots, P_k , such that for $i \in [k]$, P_i is a path from x_i to y_i and for $i \neq j$, P_i does not contain a vertex adjacent to a vertex in P_j ? |

Throughout the remainder of this section, we refer to the vertices $\{x_i, y_i\}$, where $i \in [k]$ as the *terminals* and we denote the set of all terminals by $X := \bigcup_{i \in [k]} \{x_i, y_i\}$. We furthermore use the following notation: We denote by $\text{cc}(G)$ the set of all connected components of G and for a vertex $v \in V(G)$, $C_G(v)$ refers to the connected component containing v .

We observe how a solution $\mathcal{P} = (\mathcal{P}_1, \dots, \mathcal{P}_k)$ interacts with the graph $G[V_t \cup \text{bd}(\overline{V}_t)]$, for some $t \in V(T)$. Recall that for each $i \in [k]$, \mathcal{P}_i is an (x_i, y_i) -path and additionally for $j \neq i$, there is no vertex in \mathcal{P}_i adjacent to a vertex of \mathcal{P}_j . The intersection of \mathcal{P} with $G[V_t \cup \text{bd}(\overline{V}_t)]$ is a subgraph $\mathcal{I} = (\mathcal{I}_1, \dots, \mathcal{I}_k)$, where each \mathcal{I}_i is a (possibly empty) induced disjoint union of paths which is the intersection of the

(x_i, y_i) -path \mathcal{P}_i with $G[V_t \cup \text{bd}(\bar{V}_t)]$. Note that each terminal $v_i \in \{x_i, y_i\}$ that is contained in $V_t \cup \text{bd}(\bar{V}_t)$ is also contained in $V(\mathcal{I}_i)$.

Again our goal is to bound the number of table entries at each node $t \in V(T)$ by $n^{\mathcal{O}(w)}$, so we focus on the intersection of \mathcal{I} with the crossing graph $G_{t, \bar{t}}$. There are several reasons why \mathcal{I}_i can have a nonempty intersection with the crossing graph $G_{t, \bar{t}}$: If precisely one of x_i and y_i is contained in V_t , then the path \mathcal{P}_i must cross the boundary of G_t . If both x_i and y_i are contained in V_t (\bar{V}_t), yet \mathcal{P}_i uses a vertex of \bar{V}_t (V_t), then it crosses the boundary of G_t .

We now turn to the definition of the table indices. Let us first point out what table indices in the resulting algorithm for INDUCED DISJOINT PATHS have in common with the indices in the algorithm for LONGEST INDUCED PATH and we refer to Section 8.2.1 for the motivation and details. These similarities arise since in both problems, the intersection of a solution with a crossing graph $G_{t, \bar{t}}$ is an induced disjoint union of paths.

- The intersection of \mathcal{I} with the edges of $G_{t, \bar{t}}$ is S , an induced disjoint union of paths where each component contains at least one edge.
- M is a minimal vertex cover of $G_{t, \bar{t}} - V(S)$ such that $M \cap V(S) = \emptyset$.

The first important observation to be made is that by Lemma 8.4, the number of components of S is linearly bounded in w and hence at most $\mathcal{O}(w)$ paths of \mathcal{P} can have a nonempty intersection with $G_{t, \bar{t}}$. We need to store information about which path \mathcal{P}_i (resp., to which \mathcal{I}_i) the components of S correspond to. To do so, another part of the index will be a labeling function $\lambda: \text{cc}(S) \rightarrow [k]$, whose purpose is to indicate that each component $C \in \text{cc}(S)$ is contained in $\mathcal{I}_{\lambda(C)}$. Remark that we just observed that each such λ contains at most $\mathcal{O}(w)$ entries.

Let $i \in [k]$. Again, we need to indicate how (some of) the components of S are connected via \mathcal{I}_i in $G[V_t]$. As before, we do so by considering a pairing of the vertices in $S[V_t]$ that have degree one in S , however in this case we also have to take into account the labeling function λ . That is, two such vertices s and t can only be paired if they belong to the same induced disjoint union of paths \mathcal{I}_i .

In accordance with the above discussion, we define the table entries as follows. We let $\text{tab}[t, (S, M, \lambda, Q_\lambda)] = 1$ if and only if the following conditions are satisfied.

- (i) There is an induced disjoint union of paths $\mathcal{I} = (\mathcal{I}_1, \dots, \mathcal{I}_k)$ in $G[V_t \cup \text{bd}(\bar{V}_t)]$, such that for $i \neq j$, \mathcal{I}_i does not contain a vertex adjacent (in G) to a vertex in \mathcal{I}_j . For each $i \in [k]$, we have that if $v_i \in \{x_i, y_i\} \cap V_t$, then $v_i \in V(\mathcal{I}_i)$. Furthermore, v_i has degree one in \mathcal{I}_i .
- (ii) (a) $E(\mathcal{I}) \cap E(G_{t, \bar{t}}) = E(S)$.
(b) $\lambda: \text{cc}(S) \rightarrow [k]$ is a labeling function of the connected components of S , such that for each component $C \in \text{cc}(S)$, $\lambda(C) = i$ if and only if $C \subseteq \mathcal{I}_i$.

- (iii) M is a minimal vertex cover of $G_{t,\bar{t}} - V(S)$ such that $V(\mathcal{I}) \cap M = \emptyset$.
- (iv) Let D denote the set of vertices in $S[V_t]$ that have degree one in S and let $X_t = X \cap V_t$. Then, Q_λ is a pairing (i.e. a partition into pairs) of the vertices in $D \Delta X_t$ with the following properties.⁴
 - (a) $(s, t) \in Q_\lambda$ if and only if there is a path from s to t in $\mathcal{I}[V_t]$. Note that this implies that $(s, t) \in Q_\lambda$ only if both s and t belong to the same \mathcal{I}_i for some $i \in [k]$ and in particular only if $s, t \in V(\lambda^{-1}(i)) \cup \{x_i, y_i\}$.
 - (b) For each $i \in [k]$, $(x_i, y_i) \in Q_\lambda$ only if $\lambda^{-1}(i) = \emptyset$, i.e. no component of S has label i .

We now show that the answer to the problem can be obtained from inspecting the table entries stored in the root of T .

Proposition 8.10. *G contains vertex-disjoint induced paths $\mathcal{P}_1, \dots, \mathcal{P}_k$, where \mathcal{P}_i is an (x_i, y_i) -path for each $i \in [k]$ and for $j \neq i$, no vertex in \mathcal{P}_i is adjacent to a vertex in \mathcal{P}_j , if and only if $\text{tab}[r, (\emptyset, \emptyset, \emptyset, Q_\emptyset)] = 1$, where $Q_\emptyset = \{(x_1, y_1), \dots, (x_k, y_k)\}$.*

Proof. Suppose that $\mathcal{P} = (\mathcal{P}_1, \dots, \mathcal{P}_k)$ is a solution to INDUCED DISJOINT PATHS, i.e. \mathcal{P} satisfies the conditions of the proposition. First note that since $V_r = V(G)$ and $\text{bd}(\overline{V_r}) = \emptyset$, $G[V_r \cup \text{bd}(\overline{V_r})] = G$. This implies that \mathcal{P} satisfies the conditions stated in Part (i) of the definition of a table entry being set to 1. (Clearly, each terminal $v_i \in \{x_i, y_i\}$ is contained in $V_r = V(G)$ and has degree one in \mathcal{P}_i .) Since $\text{bd}(\overline{V_r}) = \emptyset$, it follows that $G_{r,\bar{r}} = \emptyset$ and hence, $S = \emptyset$, $M = \emptyset$ and $\lambda = \emptyset$. Since each \mathcal{P}_i is an (induced) (x_i, y_i) -path and Q_\emptyset pairs terminals (x_i, y_i) for each $i \in [k]$ by assumption, Q_\emptyset satisfies Part (iv) of the definition of a table entry being set to 1. Remark that Part (iv.b) is satisfied since $\lambda = \emptyset$ and hence $\lambda^{-1}(i) = \emptyset$ for all $i \in [k]$. It follows that $\text{tab}[r, (\emptyset, \emptyset, \emptyset, Q_\emptyset)] = 1$.

Now suppose that $\text{tab}[r, (\emptyset, \emptyset, \emptyset, Q_\emptyset)] = 1$. Then by Part (i) of the definition of the table entries, there is an induced disjoint union of paths $\mathcal{I} = (\mathcal{I}_1, \dots, \mathcal{I}_k)$ in $G[V_r \cup \text{bd}(\overline{V_r})] = G$ such that for each $i \neq j$, no vertex in \mathcal{I}_i is adjacent to a vertex in \mathcal{I}_j . Furthermore, for each $i \in [k]$, $\{x_i, y_i\} \subseteq V(\mathcal{I}_i)$, and both x_i and y_i have degree one in \mathcal{I}_i . Since Q_\emptyset pairs all (x_i, y_i) , Part (iv) allows us to conclude that each \mathcal{I}_i is an (x_i, y_i) -path. \square

Again, we denote by \mathcal{S}_t the set of all induced unions of paths on at most $4w$ vertices in $G_{t,\bar{t}}$, for $S \in \mathcal{S}_t$ and by $\mathcal{M}_{t,S}$ the set of all minimal vertex covers of $G_{t,\bar{t}} - V(S)$. We let Λ_S denote the set of all labeling functions of the connected components of S and for $\lambda \in \Lambda_S$ by $Q_{t,S,\lambda}$ the set of all pairings in accordance with Part (iv) of the table definition.

⁴We denote by ‘ Δ ’ the symmetric difference, i.e. $D \Delta X_t = (D \cup X_t) \setminus (D \cap X_t)$. Q_λ is a pairing on $D \Delta X_t$, since if a terminal v_i is contained in D , it is supposed to be paired ‘with itself’: Since v_i has degree one in \mathcal{I}_i by (i) and is incident to an edge in S , v_i cannot be paired with another vertex.

Proposition 8.11. *For each $t \in V(T)$, there are at most $n^{\mathcal{O}(w)}$ table entries in tab_t and they can be enumerated in time $n^{\mathcal{O}(w)}$.*

Proof. By the proof of Proposition 8.7, we know that $|\mathcal{S}_t| \leq n^{\mathcal{O}(w)}$ and $|\mathcal{M}_{t,S}| \leq \mathcal{O}(n^{4w})$. Since Lemma 8.4 also bounds the number of connected components in $S \in \mathcal{S}_t$ by $\mathcal{O}(w)$, we can conclude that $|\Lambda_S| \leq k^{\mathcal{O}(w)}$, since then for each $\lambda \in \Lambda_S$, $\lambda^{-1}(i) \neq \emptyset$ for at most $\mathcal{O}(w)$ values of $i \in [k]$. The same reasoning can be applied to count $|\mathcal{Q}_{t,S,\lambda}|$, since each $j \in [k]$ such that $\lambda^{-1}(j) = \emptyset$ allows no further choices for pairings in $\mathcal{Q}_{t,S,\lambda}$: Either both x_j and y_j are contained in V_t and we pair them, or neither of them is contained in V_t and we disregard them. We can conclude that $|\mathcal{Q}_{t,S,\lambda}| \leq w^{\mathcal{O}(w)}$. To summarize, there are at most

$$\mathcal{O}(n^{4w}) \cdot n^{\mathcal{O}(w)} \cdot k^{\mathcal{O}(w)} \cdot w^{\mathcal{O}(w)} = n^{\mathcal{O}(w)}$$

entries in tab_t for each $t \in V(T)$. Note that even if $k = \mathcal{O}(n)$, we obtain a bound of $n^{\mathcal{O}(w)}$ on the number of table entries stored at each node $t \in V(T)$. The time bound on the enumeration of the indices follows from the same argument given in the proof of Proposition 8.7. \square

We now explain how the table entries tab_t are computed for each $t \in V(T)$.

Leaves of T . Let $t \in V(T)$ be a leaf of T and $v = \mathcal{L}^{-1}(t)$. Any nonempty intersection of an induced disjoint union of paths with $G_{t,\bar{t}}$ is either a single edge using v or a path on two edges having v as the middle vertex. Hence, all nonempty S such that the corresponding table entry is to 1 are either a single edge using v , and we denote this set by $\mathcal{S}_{t,v}^1$ or a path on two edges with v as the middle vertex and we denote this set by $\mathcal{S}_{t,v}^2$.

Suppose $S \in \mathcal{S}_{t,v}^1$. Since $G_{t,\bar{t}} - V(S)$ has no edges, $M = \emptyset$. If a solution intersects S , this means that v is a terminal vertex, i.e. $v \in \{x_{i^*}, y_{i^*}\}$ for some $i^* \in [k]$. Hence, (with a slight abuse of notation) $\lambda(S) = i^*$ and since v is the only degree one vertex in $S[V_t]$, $Q_\lambda = \emptyset$.

Now suppose $S \in \mathcal{S}_{t,v}^2$. First we observe that in this case, v cannot be a terminal vertex, i.e. $v \notin X$. Again, $G_{t,\bar{t}} - V(S)$ has no edges, so $M = \emptyset$. Let u_1, u_2 be the vertices in S other than v . We distinguish the following cases.

Case L1 (no terminal). If neither of u_1 and u_2 is a terminal vertex, then S can be a subgraph of any (x_i, y_i) -path, so we allow all choices of $\lambda(S) = i$ for $i \in [k]$.

Case L2 (one terminal). If precisely one of u_1 and u_2 is a terminal vertex for some $i^* \in [k]$, then the only choice for λ is such that $\lambda(S) = i^*$.

Case L3 (two terminals, same path). If $\{u_1, u_2\} = \{x_{i^*}, y_{i^*}\}$ for some $i^* \in [k]$, then the only choice for λ is such that $\lambda(S) = i^*$. Note that if $u_1 \in \{x_i, y_i\}$ and $u_2 \in \{x_j, y_j\}$ for $i \neq j$, then S is not a valid partial solution.

In all of the above cases, there is no degree one vertex in $S[V_t]$, so $Q_\lambda = \emptyset$.

Eventually, we have to consider the cases when no edge of $G_{t,\bar{t}}$ is used in a partial solution. Note that this is only possible if v is not a terminal vertex. The choices for the minimal vertex covers are then $\{v\}$ and $N(v)$ and since $S = \emptyset$, it follows that $\lambda = \emptyset$ and $Q_\lambda = \emptyset$.

To summarize, we set the table entries in tab_t as follows. We let:

$$\text{tab}[t, (S, M, \lambda, Q_\lambda)] := \begin{cases} 1, & \text{if } S \in \mathcal{S}_{t,v}^1, M = \emptyset, \lambda(S) = i^*, Q_\lambda = \emptyset \text{ and } v \in \{x_{i^*}, y_{i^*}\} \\ 1, & \text{if } S \in \mathcal{S}_{t,v}^2, M = \emptyset, \lambda(S) = i \text{ for some } i \in [k], Q_\lambda = \emptyset \\ & \text{and } \{v, u_1, u_2\} \cap X = \emptyset \text{ (Case L1)} \\ 1, & \text{if } S \in \mathcal{S}_{t,v}^2, M = \emptyset, \lambda(S) = i^*, Q_\lambda = \emptyset \\ & \text{and } u_{j_1} \in \{x_{i^*}, y_{i^*}\}, v \notin X, u_{j_2} \notin X \text{ where } \{j_1, j_2\} = [2] \text{ (Case L2)} \\ 1, & \text{if } S \in \mathcal{S}_{t,v}^2, M = \emptyset, \lambda(S) = i^*, Q_\lambda = \emptyset \\ & \text{and } \{u_1, u_2\} = \{x_{i^*}, y_{i^*}\}, v \notin X \text{ (Case L3)} \\ 1, & \text{if } S = \emptyset, M \in \{\{v\}, N(v)\}, \lambda = \emptyset, Q_\lambda = \emptyset \text{ and } v \notin X \\ 0, & \text{otherwise} \end{cases}$$

Internal nodes of T . We argue that for each internal node $t \in V(T)$ and each index $\mathcal{I}_t := (S, M, \lambda, Q_\lambda) \in \mathcal{S}_t \times \mathcal{M}_{t,S} \times \Lambda_S \times \mathcal{Q}_{t,S,\lambda}$, the value of the table entry $\text{tab}[\mathcal{I}_t]$ can be computed in complete analogy with the algorithm for LONGEST INDUCED PATH (LIP). In particular, it can be viewed as performing for each $i \in [k]$ the steps presented in the algorithm of LIP. Some attention must be devoted to guaranteeing that there is no edge between two components labeled i and j (where $i \neq j$). However, as outlined below, partial solutions to the two problems represented by their respective indices have almost exactly the same structure and in that sense, the routine presented for LIP already captures this requirement. More precisely, the only way in which unwanted edges could be added during the join in INDUCED DISJOINT PATHS (IDP) is if they cross the boundary. By definition, there will never be any unwanted edges in the intersection of a partial solution with the boundary in both problems. Subsequently, the only place where they can appear is between intermediate vertices of a combined partial solution. In the join of LIP, we explicitly ensure that no edges between intermediate vertices appear (via the minimal vertex cover) and in the same way we can enforce in IDP that no edges appear between intermediate vertices of components labeled i and j (for $i \neq j$) when combining two partial solutions.

We now argue in detail the similarity between the partial solution based on the definitions of the table entries for the respective problems. Throughout the following, we refer to the conditions stated in the definition of the table entries for LONGEST INDUCED PATH by LIP(\cdot) and for INDUCED DISJOINT PATHS by IDP(\cdot).

First, observe that in both cases, the intersection of a solution with the graph $G[V_t \cup \text{bd}(\overline{V}_t)]$ is an induced disjoint union of paths \mathcal{I} . The key difference between

the two problems is that in LIP we are interested in the solution size, whereas in IDP we are not and that in IDP we additionally have to take into account to which (x_i, y_i) -path the components of \mathcal{I} belong. Note that aside from the size constraint, LIP(i) and IDP(i) express precisely the same thing if $k = 1$ in IDP and the entry j in LIP takes the role of the fixed terminals in IDP.

LIP(ii) and IDP(ii.a) are in fact identical, in particular this means that the intersection S of a partial solution with the crossing graph $G_{t,\bar{t}}$ is the same in both problems. So to make the join procedure described for LIP work for IDP in a way that it preserves Condition IDP(ii), we only have to take into account the behavior of the labeling function λ introduced in IDP(ii.b). Note that again, if $k = 1$ in IDP, then LIP(ii) and IDP(ii) express precisely the same thing. LIP(iii) and IDP(iii) are as well identical, so all parts of the algorithm for LIP that ensure that this condition holds can immediately applied (and argued for) in the join of IDP.

It remains to argue the similarity of LIP(iv) and IDP(iv). Since we do not know the endpoints of the induced path we are looking for in LIP, whereas in IDP we do, slightly differing languages were used to express the properties of the pairings. In LIP(iv), two vertices are paired if contracting all edges in $\mathcal{I} - E(G_{t,\bar{t}})$ leaves an edge between the two vertices in the graph. But this happens precisely when there is a path between these two vertices in $\mathcal{I}[V_t]$, the condition stated in IDP(iv.a). The remainder of IDP(iv) is devoted to including the terminals in the pairings, so the necessary modifications in the algorithm of LIP to work for IDP such that it respects IDP(iv) are straightforward. Note that again if $k = 1$ in IDP, LIP(iv) and IDP(iv) express the same thing where j takes the role of the terminals contained in V_t .

By the above discussion, the fact that we enumerate all possible solutions in the leaves of T , Propositions 8.10 and 8.11 and in the light of the correctness (and runtime) of the join of the algorithm for LONGEST INDUCED PATH, we have the following theorem.

Theorem 8.12. *There is an algorithm that given a graph G on n vertices, pairs of terminal vertices $(x_1, y_1), \dots, (x_k, y_k)$ and a branch decomposition (T, \mathcal{L}) of G , solves INDUCED DISJOINT PATHS in time $n^{\mathcal{O}(w)}$, where w denotes the mim-width of (T, \mathcal{L}) .*

8.2.3 H -Induced Topological Minor

Let G be a graph and $uv \in E(G)$. We call the operation of replacing the edge uv by a new vertex x and edges ux and xv the *edge subdivision* of uv . We call a graph H a *subdivision* of G if it can be obtained from G by a series of edge subdivisions. We call H an *induced topological minor* of G if a subdivision of H is isomorphic to an induced subgraph of G .

H-INDUCED TOPOLOGICAL MINOR parameterized by $w := \text{mimw}(T, \mathcal{L})$

Input: A graph G with branch decomposition (T, \mathcal{L})

Question: Does G contain H as an induced topological minor?

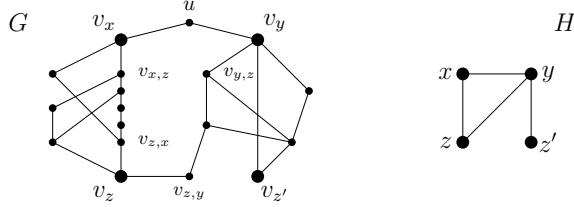


Figure 8.4: An example YES-instance of H -INDUCED TOPOLOGICAL MINOR. Note that $v_{y,z'} = v_{z'}$ and $v_{z',y} = v_y$ since $\{v_y, v_{z'}\} \in E(G)$ in accordance with Step 2(a). For each of the remaining edges in H there are precisely two corresponding edges each in G , illustrating Step 2(b). Furthermore, $u = v_{x,y} = v_{y,x}$, illustrating the special case dealt with in Step 2(c).

Theorem 8.13. *There is an algorithm that given a graph G on n vertices and a branch decomposition (T, \mathcal{L}) of G , solves H -INDUCED TOPOLOGICAL MINOR in time $n^{\mathcal{O}(w)}$, where H is a fixed graph and w the mim-width of (T, \mathcal{L}) .*

Proof. Let H be a fixed graph. To solve H -INDUCED TOPOLOGICAL MINOR, we have to find a map φ from the vertices of H to a set of vertices in G such that there is an edge in $\{x, y\}$ if and only if there is an induced path between $\varphi(x)$ and $\varphi(y)$ in G such that additionally, for each pair P_1, P_2 of such paths, no vertex in P_1 is adjacent to a vertex in P_2 . We do so by guessing to which vertices in G the vertices of H are mapped and after some preprocessing, we run the algorithm for INDUCED DISJOINT PATHS with (at most) $|E(H)|$ pairs of terminals to find the induced paths in G corresponding to the edges in H (see Figure 8.4).

Step 1 (Branching). For each vertex $x \in V(H)$, we guess the corresponding vertex $v_x \in V(G)$ of x , i.e. $\varphi(x) = v_x$. Hence we require that for all $x, y \in V(H)$, $x = y \Leftrightarrow v_x = v_y$ and that $\deg_G(v_x) \geq \deg_H(x)$. We let $X := \bigcup_{x \in V(H)} v_x$. Furthermore, for each $x \in V(H)$ and neighbor y of x , we guess a distinct corresponding neighbor $v_{x,y}$ in G . Note that since $\deg_G(v_x) \geq \deg_H(x)$ for all $x \in V(H)$, there are sufficiently many such vertices in G to choose from. The vertex $v_{x,y}$ is the vertex adjacent to v_x in the induced path in G corresponding to the edge $\{x, y\}$ in H . We let $Y_x := \bigcup_{y \in N_H(x)} v_{x,y}$ and $Y := \bigcup_{x \in V(H)} Y_x$ and we refer to the vertices in Y as the *neighbor vertices*. We branch on each such choice of $X \cup Y$.

Step 2 (Preprocessing according to $X \cup Y$). In this step we check whether the current choice of X and Y is valid and prepare the current instance for the call to the algorithm of INDUCED DISJOINT PATHS.

- (a). For each edge $\{x, y\} \in E(H)$, if $\{v_x, v_y\} \in E(G)$, then we remove the edge $\{x, y\}$ from H without changing the answer to the problem: $\{v_x, v_y\}$ is the induced path in G corresponding to the edge $\{x, y\}$. If on the other hand, $\{x, y\} \notin E(H)$ but $\{v_x, v_y\} \in E(G)$, then we discard this choice of $X \cup Y$.

- (b). We check whether for each edge $\{x, y\} \in E(H)$ there are precisely two corresponding edges, namely $\{v_x, v_{x,y}\}$ and $\{v_y, v_{y,x}\}$ in G and that $G[X \cup Y]$ induces no further edges. In particular, if $G[X \cup Y]$ does contain any additional edges, we discard this choice of X and Y .
- (c). If there is a vertex $u \in V(G)$ which is chosen more than once as a neighbor vertex, i.e. there exists a set $A \subseteq V(H)$ such that $u \in \bigcap_{a \in A} Y_a$ with $|A| > 1$, then we proceed only if $A = \{x, y\}$ for some $x, y \in V(H)$ and such that $u = v_{x,y} = v_{y,x}$. In that case, we remove the edge $\{x, y\}$ from H without changing the answer to the problem: The set $\{v_x, u, v_y\}$ induces the path in G corresponding to the edge $\{x, y\}$. Furthermore, we remove the vertices in $N(u) \setminus X$ from G , since these vertices cannot be used by any other path corresponding to an edge in H .

Step 3 (Execution of IDP-algorithm). We run the algorithm for INDUCED DISJOINT PATHS on $G - X$ with pairs of terminals $(v_{x,y}, v_{y,x})_{\{x,y\} \in E(H)}$. (Note that some edges of H might have been removed in Steps 2(a) and 2(c) and that some vertices of G might have been removed in Step 2(c).)

Step 4 (Return). We let the algorithm return YES if at least one run of INDUCED DISJOINT PATHS in Step 3 returned YES, and NO otherwise.

We now analyze the runtime of the algorithm. In Step 1, we branch in $n^{\mathcal{O}(|E(H)|)}$ ways, the number of choices for the sets X and Y . The checks in Step 2 can be performed in time polynomial in n and each execution of the algorithm for INDUCED DISJOINT PATHS in Step 3 takes time $n^{\mathcal{O}(w)}$ by Theorem 8.12. So the total runtime of the algorithm is $n^{\mathcal{O}(|E(H)|)} \cdot n^{\mathcal{O}(w)} = n^{\mathcal{O}(w)}$, as H is fixed. \square

8.3 Feedback Vertex Set on Mim-Width

A feedback vertex set in a graph is a subset of its vertex set whose removal results in an acyclic graph. The problem of finding a smallest such set is one of Karp's 21 famous NP-complete problems [179] and many algorithmic techniques have been developed to attack this problem, see e.g. the survey [122]. The study of FEEDBACK VERTEX SET through the lens of parameterized algorithmics dates back to the earliest days of the field [103] and throughout the years numerous efforts have been made to obtain faster algorithms for this problem [28, 74, 94, 99, 103, 104, 149, 177, 247, 248]. In terms of parameterizations by structural properties of the graph, FEEDBACK VERTEX SET is known to be FPT parameterized by treewidth [94] and clique-width [59], and W[1]-hard but in XP parameterized by the size of an independent set and the size of a maximum induced matching [172].

In this chapter, we study FEEDBACK VERTEX SET parameterized by mim-width, formally defined as follows.

FEEDBACK VERTEX SET parameterized by $w := \text{mimw}(T, \mathcal{L})$

Input: Graph G and one of its rooted branch decompositions (T, \mathcal{L}) , integer k

Question: Does G contain a feedback vertex set of size at most k ?

We give an XP-time algorithm for this parameterized problem, assuming that a branch decomposition of bounded mim-width is given. This problem was mentioned as an ‘interesting topic for further research’ in [172]. Our algorithm can be applied to WEIGHTED FEEDBACK VERTEX SET as well, which on several of these classes was not known to be solvable in polynomial time.

Theorem 8.14. *Given an n -vertex graph and one of its branch decompositions of mim-width w , we can solve (WEIGHTED) FEEDBACK VERTEX SET in time $n^{\mathcal{O}(w)}$.*

Let us explain some of the essential ingredients of our dynamic programming algorithm which solves the dual MAXIMUM (WEIGHT) INDUCED FOREST problem. Note that the two problems are equivalent in the mim-width parameterization. A crucial observation is that if a forest contains no induced matching of size $w+1$, then the number of internal vertices of the forest is bounded by $6w$ (Lemma 8.16). Motivated by this observation, given a forest, we define the forest obtained by removing its isolated vertices and leaves to be its *reduced forest*. Let (A, B) be a cut of a graph G and denote by $G_{A,B}$ the bipartite graph induced by this cut. The observation implies that if there is no induced matching of size $w+1$ in $G_{A,B}$, then there are at most $\mathcal{O}(n^{6w})$ possible reduced forests of some induced forests consisting of edges crossing this cut. We enumerate all of them, and use them as indices of the table of our algorithm.

Following the given branch decomposition, we want to recursively ask whether for a forest R in $G_{A,B}$, there is a forest in the graph on the union of A and the boundary⁵ of B , such that its restriction to $G_{A,B}$ has R as a reduced forest. When we decide to add some other vertex from B to our forest at a later stage of the algorithm, we do not want to have an edge from A to B not intersecting the vertices of R . One way to avoid these additional edges is to take a vertex cover M of $G_{A,B} - V(R)$, and then ask whether there is a forest F on the union of A and the boundary of B such that it avoids M and $G_{A,B} \cap F$ has R as a reduced forest. We observe that for any such forest F , there is always a vertex cover M that satisfies this condition. This suggests that we add all possible minimal vertex covers of $G_{A,B} - V(R)$ as a second component of the table indices.

To argue that the number of table entries stays bounded by $n^{\mathcal{O}(w)}$, we furthermore make use the Minimal Vertex Covers Lemma (Corollary 8.3) which states that every n -vertex bipartite graph with maximum induced matching size w has at most n^w minimal vertex covers, and that we can enumerate them within the same time bound.

⁵I.e. the vertices in B that have neighbors in A .

Additionally, we observe that our algorithm can also be applied to the connected variant of the problem, i.e. it can be used to solve the MAXIMUM (WEIGHT) INDUCED TREE problem in the same parameterization and time bound as well.

8.3.1 Reduced Forests and Vertex Covers

In this section, we introduce some technical concepts and prove some technical lemmas that will be used to devise and analyze the FEEDBACK VERTEX SET algorithm given in Section 8.3.2. As alluded to above, we would like to store subgraphs of the intersection of induced forests with the edges crossing a cut. We call these subgraphs *reduced forests* and we begin by defining them formally.

Definition 8.15 (Reduced Forest). Let F be a forest. A *reduced forest* of F is an induced subforest of F obtained as follows.

- (i) Remove all isolated vertices of F .
- (ii) For each component C of F with $|V(C)| = 2$, remove one of its vertices.
- (iii) For each component C of F with $|V(C)| \geq 3$, remove all leaves of C .

Note that if F has no single-edge component, then the reduced forest is uniquely defined. We give an upper bound on the size of a reduced forest of a forest F by a function of the size of a maximum induced matching in F .

Lemma 8.16. *Let p be a positive integer. If F is a forest whose maximum induced matching has size at most p and \mathfrak{R} is a reduced forest of F , then $|V(\mathfrak{R})| \leq 6p$.*

Proof. For a forest F , we denote by $m(F)$ the size of a maximum induced matching in F . We prove the lemma by induction on $m(F)$. If $m(F) = 0$, then F contains no edges, and $|V(\mathfrak{R})| = 0$. If $m(F) = 1$, then F consists of one component that contains no path of length 4 and (possibly) some isolated vertices which implies that \mathfrak{R} contains at most 2 vertices. We may assume that $m(F) = p > 1$. We may further assume that F contains no isolated vertices, as they will be removed in the reduced forest.

Suppose F contains a connected component C containing no path of length 4. As observed, C contains no induced matching of size larger than one. Since C contains an edge, we have $m(F - V(C)) = m(F) - 1$. Let $\mathfrak{R}_{F-V(C)}$ be a reduced forest of $F - V(C)$ that is a restriction of \mathfrak{R} . By the induction hypothesis, $\mathfrak{R}_{F-V(C)}$ contains at most $6(p - 1)$ vertices, and we have that \mathfrak{R} contains at most $6(p - 1) + 2 \leq 6p$ vertices. Thus, we may assume that every connected component C of F contains a path of length 4, implying that its reduced forest contains at least 3 vertices. It also implies that every connected component of F has a unique reduced forest.

Now, suppose F contains a path $v_1v_2v_3v_4v_5$ such that v_1 and v_5 are not leaves of F , and v_2, v_3, v_4 have degree 2 in \mathfrak{R} . Let F' be the forest obtained from F by removing v_2, v_3, v_4 and adding an edge v_1v_5 . Let \mathfrak{R}' be the reduced forest of F' .

We claim that $m(F') \leq m(F) - 1$. Let M be a maximum induced matching of F' . If M contains the edge v_1v_5 , then we can obtain an induced matching for F by removing v_1v_5 and adding v_1v_2 and v_4v_5 . If M does not contain v_1v_5 , then one of v_1 and v_5 is not matched by M . Then for F , we can select one of v_2v_3 and v_3v_4 to increase the size of an induced matching. Thus, we have $m(F') \leq m(F) - 1$. By the induction hypothesis, \mathfrak{R}' contains at most $6(p - 1)$ vertices, and thus \mathfrak{R} contains at most $6(p - 1) + 3 = 6p - 3$ vertices. We may assume that there is no such path.

Let C be a connected component of F , and \mathfrak{R}_C be the reduced forest of C . As \mathfrak{R}_C contains at least 3 vertices, the leaves of \mathfrak{R}_C form an independent set. Let t be the number of leaves in \mathfrak{R}_C . Since each leaf of \mathfrak{R}_C is adjacent to a leaf of C , C contains an induced matching of size at least t . Thus, $m(F - V(C)) \leq m(F) - t$. Note that \mathfrak{R}_C contains at most t vertices of degree at least 3. Also, by the previous argument, there are at most 2 vertices between two vertices of degree other than 2 in \mathfrak{R}_C . Thus, \mathfrak{R}_C contains at most $t + t + 2(2t - 1) \leq 6t$ vertices. Therefore, the result follows by the induction hypothesis. \square

Let (A, B) be a vertex partition of a graph G , R be some induced forest in $G_{A,B}$, and M a minimal vertex cover of $G_{A,B} - V(R)$. In the algorithm, we want to ask if there exists an induced forest F in $G[A \cup \text{bd}(B)]$ such that R is a reduced forest of $F \cap G_{A,B}$ and F avoids the vertices in M . However, it turns out that in this direct formulation it is difficult to account for edges between vertices in $\text{bd}(B)$. We therefore define the following notion on an induced forest in $G[A \cup \text{bd}(B)] - E(G[\text{bd}(B)])$, instead of $G[A \cup \text{bd}(B)]$.

Definition 8.17 (Respecting Forest). Let (A, B) be a vertex partition of a graph G . Let R be an induced forest in $G_{A,B}$ and M be a minimal vertex cover of $G_{A,B} - V(R)$. An induced forest F in $G[A \cup \text{bd}(B)] - E(G[\text{bd}(B)])$ respects (R, M) if it satisfies the following.

- (i) R is a reduced forest of $G_{A,B} \cap F$.
- (ii) $V(F) \cap M = \emptyset$.

Suppose R is an induced forest in $G_{A,B}$. For an induced forest F of G containing $V(R)$, there are two necessary conditions for R to be a reduced forest of $F \cap G_{A,B}$. First, in $F \cap G_{A,B}$, every vertex in $V(F \cap G_{A,B}) \setminus V(R)$ has at most one neighbor in R ; otherwise, when we take a reduced forest of $F \cap G_{A,B}$, this vertex should remain. Second, in $F \cap G_{A,B}$, every leaf x of R should have a neighbor y in $V(F \cap G_{A,B}) \setminus V(R)$ (and the only neighbor of y in R is x); otherwise, we would have removed x when taking a reduced forest.

Motivated by this observation we define the notion of potential leaves, which is a possible leaf neighbor of some vertex in $V(R)$. See Figure 8.5 for an illustration.

Definition 8.18 (Potential Leaves). Let (A, B) be a vertex partition of a graph G . Let R be an induced forest in $H := G_{A,B}$ and M be a minimal vertex cover of

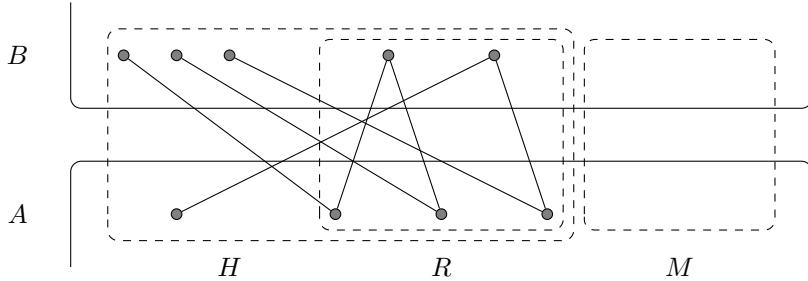


Figure 8.5: The graph R is an induced forest of $G_{A,B}$ and M is a minimal vertex cover of $G_{A,B} - V(R)$. Observe that R is a reduced forest of H . The four vertices in $V(H) \setminus V(R)$ are potential leaves with respect to R and M .

$H - V(R)$. For each vertex $x \in V(R)$, we define its set of *potential leaves* with respect to R and M , denoted by $PL_{R,M}(x)$, as

$$PL_{R,M}(x) := N_H(x) \setminus (N_H(V(R) \setminus \{x\}) \cup (M \cup V(R))) .$$

We can observe the following.

Observation 8.19. Every forest F respecting (R, M) contains at least one vertex in $PL_{R,M}(x)$ for each leaf x of R .

For a subset A' of A , we consider a pair of an induced forest R' and a minimal vertex cover M' of $G_{A',V(G)\setminus A'} - V(R')$ and we say that this pair is a *restriction* of a pair of R and M for A , if they satisfy certain natural properties. In the dynamic programming algorithm, we use this notion to study the structure of partial solutions w.r.t. cuts corresponding to a node t and the children of t .

Definition 8.20 (Restriction). Let (A_1, A_2, B) be a vertex partition of a graph G . Let R be an induced forest in $G_{A_1 \cup A_2, B}$ and M be a minimal vertex cover of $G_{A_1 \cup A_2, B} - V(R)$. An induced forest R_1 in $G_{A_1, A_2 \cup B}$ and a minimal vertex cover M_1 of $G_{A_1, A_2 \cup B} - V(R_1)$ are *restrictions* of R and M to $G_{A_1, A_2 \cup B}$ if they satisfy the following:

- (i) $V(R) \cap A_1 \subseteq V(R_1)$ and $V(R_1) \cap B \subseteq V(R)$.
- (ii) Every vertex in $(V(R_1) \setminus V(R)) \cap A_1$ has at most one neighbor in $V(R) \cap B$.
- (iii) $M \cap A_1 \subseteq M_1$ and $M_1 \cap B \subseteq M$.

Lastly, we define a notion of compatibility of two partial solutions for the algorithm. To clarify, in the following definition, the partitions of the connected components of R_i represent connectivity information about induced forests in $G[A_i \cup \text{bd}(A_{3-i} \cup B)] - E(G[\text{bd}(A_{3-i} \cup B)])$ respecting R_i .

Definition 8.21 (Compatibility). Let (A_1, A_2, B) be a vertex partition of a graph G . Let R be an induced forest in $G_{A_1 \cup A_2, B}$, and for each $i \in \{1, 2\}$, let R_i be an induced forest in $G_{A_i, A_{3-i} \cup B}$, and P_i be a partition of $\mathcal{C}(R_i)$. We construct an auxiliary graph Q with respect to (R, R_1, R_2, P_1, P_2) in G as follows. Let Q be the graph on the vertex set $\mathcal{C}(R) \cup \mathcal{C}(R_1) \cup \mathcal{C}(R_2)$ such that

- for H_1 and H_2 in distinct sets of $\mathcal{C}(R), \mathcal{C}(R_1), \mathcal{C}(R_2)$, H_1 is adjacent to H_2 in Q if and only if $V(H_1) \cap V(H_2) \neq \emptyset$,
- for $i \in \{1, 2\}$ and the set of connected components contained in the same part of P_i , we add a path on the parts of P_i ,
- $\mathcal{C}(R)$ is an independent set.

We say that the tuple (R, R_1, R_2, P_1, P_2) is *compatible* in G if Q has no cycles. We define $\mathcal{U}(R, R_1, R_2, P_1, P_2)$ to be the partition of $\mathcal{C}(R)$ such that for $H_1, H_2 \in \mathcal{C}(R)$, H_1 and H_2 are contained in the same part if and only if they are contained in the same connected component of Q .

The remainder of this section is devoted to proving three technical propositions related to the notions introduced above that will be important to establish the correctness of the algorithm proposed in Section 8.3.2. Let $t \in V(T)$ be an internal node in the given branch decomposition of G with children a and b . In Section 8.3.1 we show that given any forest F_t in $G[V_t \cup \text{bd}(\overline{V_t})]$ respecting a pair (R_t, M_t) , we can find restrictions (R_a, M_a) and (R_b, M_b) to $G_{a, \overline{a}}$ and $G_{b, \overline{b}}$, respectively, such that a forest F_a in $G[V_a \cup \text{bd}(\overline{V_a})]$ respecting (R_a, M_a) and a forest F_b in $G[V_b \cup \text{bd}(\overline{V_b})]$ respecting (R_b, M_b) can be combined to the forest F_t , i.e. we have that $F_t = F_a \cup F_b$. In Section 8.3.1 we prove the converse direction. For the sake of generality, we will state the results in terms of a 3-partition (A_1, A_2, B) rather than $(V_a, V_b, \overline{V_t})$ (i.e., independently of a branch decomposition of a graph).

Top to Bottom

Proposition 8.22. Let (A_1, A_2, B) be a vertex partition of a graph G . Let R be an induced forest in $G_{A_1 \cup A_2, B}$ and M be a minimal vertex cover of $G_{A_1 \cup A_2, B} - V(R)$. Let H be an induced forest in $G[A_1 \cup A_2 \cup \text{bd}(B)] - E(G[\text{bd}(B)])$ respecting (R, M) .

Then there are restrictions (R_1, M_1) and (R_2, M_2) of (R, M) to $G_{A_1, A_2 \cup B}$ and $G_{A_2, A_1 \cup B}$, respectively, such that

- (i) for each $i \in \{1, 2\}$, $H \cap G[A_i \cup \text{bd}(A_{3-i} \cup B)] - E(G[\text{bd}(A_{3-i} \cup B)])$ respects (R_i, M_i) ,
- (ii) every vertex in $(V(R) \setminus (V(R_1) \cup V(R_2))) \cap B$ has at least two neighbors in $(V(R_1) \cap A_1) \cup (V(R_2) \cap A_2)$, and
- (iii) for each $i \in \{1, 2\}$, $V(R_i) \cap A_{3-i} \subseteq V(R_{3-i})$.

Proof. Let $A = A_1 \cup A_2$ and $H_{A,B} = H \cap G_{A,B}$. For each $i \in \{1, 2\}$, let $F_i^* := H \cap G[A_i \cup \text{bd}(A_{3-i} \cup B)] - E(G[\text{bd}(A_{3-i} \cup B)])$, and let $F_i := F_i^* \cap G_{A_i, A_{3-i} \cup B}$, and let R_i be a reduced forest of F_i such that the following holds.

(Single-edge Rule I.) For a single-edge component vw of F_i with $v \in V(R)$ and $w \notin V(R)$, we select v as a vertex of R_i .

(Single-edge Rule II.) For an edge vw with $v \in A_1$, $w \in A_2$, and $v, w \notin V(R)$ that is a single-edge component in both F_1 and F_2 , we select the same vertex as a vertex of R_i in both F_1 and F_2 .

We first prove (ii).

Claim 8.22.1. Every vertex in $(V(R) \setminus (V(R_1) \cup V(R_2))) \cap B$ has at least two neighbors in $(V(R_1) \cap A_1) \cup (V(R_2) \cap A_2)$.

Proof. Suppose there exists a vertex v in $(V(R) \setminus (V(R_1) \cup V(R_2))) \cap B$ having at most one neighbor in $(V(R_1) \cap A_1) \cup (V(R_2) \cap A_2)$. If $N_H(v)$ contains exactly one vertex w , then vw was a single-edge component of $H_{A,B}$; otherwise, v would have been removed while taking the reduced forest of $H_{A,B}$. But then $w \notin V(R)$ because $v \in V(R)$, and Single-edge rule I forces $v \in V(R_1) \cup V(R_2)$, a contradiction with the assumption. So v has at least two neighbors in $V(H) \cap (A_1 \cup A_2)$. Thus, v has a neighbor not contained in $(V(R_1) \cap A_1) \cup (V(R_2) \cap A_2)$. Let w be such a vertex, and without loss of generality, we assume that $w \in A_1$.

If v has a neighbor other than w in F_1 , then v is contained in R_1 . So, in F_1 , w is the unique neighbor of v in $V(H) \cap A_1$. Also, since $w \notin V(R_1)$, v is the unique neighbor of w in F_1 . Then vw is a single-edge component of F_1 , and by Single-edge Rule I, we selected v as a vertex of R_1 . This contradicts $v \notin V(R_1)$.

We conclude that every vertex in $(V(R) \setminus (V(R_1) \cup V(R_2))) \cap B$ has at least two neighbors in $(V(R_1) \cap A_1) \cup (V(R_2) \cap A_2)$. \square

We prove (iii).

Claim 8.22.2. For each $i \in \{1, 2\}$, $V(R_i) \cap A_{3-i} \subseteq V(R_{3-i})$.

Proof. Let $v \in V(R_i) \cap A_{3-i}$. As $v \in V(R_i)$, v has a neighbor w in F_i . Note that either v has at least two neighbors in F_i or vw is a single-edge component of F_i such that v is selected as a vertex of R_i .

Assume that v has at least two neighbors in F_i . By construction, edges between these two vertices and v are in H , and therefore, these two edges are also contained in F_{3-i} as well. Then since v has degree at least 2 in F_{3-i} , v is in R_{3-i} , as required.

Thus, we may assume that vw is a single-edge component of F_i . If $w \in V(R)$, then it should have a neighbor in B , which contradicts the fact that vw is a single-edge component of F_i . So, $w \notin V(R)$.

Note that vw may not be a single-edge component of F_{3-i} because of edges between A_2 and B . If $N_{F_{3-i}}(v)$ contains a vertex other than w , then v is selected

as a vertex of R_{3-i} as w is a leaf of F_{3-i} . We may assume that w is the unique neighbor of v in F_{3-i} . In particular, $v \notin V(R)$. Since v is selected as a vertex of R_i , by Single-edge Rule II, v is also selected as a vertex of R_{3-i} . Thus, $v \in V(R_{3-i})$, as required. \square

In the remainder of this proof we show (i), i.e. that for each $i \in \{1, 2\}$, R_i is a restriction of R . We will construct a minimal vertex cover M_i later, after confirming first two conditions of Definition 8.20. We give the proof for $i = 1$; an analogous proof holds for $i = 2$.

Claim 8.22.3. $V(R) \cap A_1 \subseteq V(R_1)$.

Proof. Let $v \in V(R) \cap A_1$. Since $v \in V(R)$, v has at least one neighbor in $H_{A,B}$, and thus, v has at least one neighbor in F_1 on B as well. So, either v has degree at least 2 in F_1 or the unique neighbor of v in F_1 is its potential leaf with respect to (R, M) in $H_{A,B}$. In the former case, clearly v is contained in R_1 , and in the latter case, v was chosen as a vertex of R_1 by Single-edge Rule I. \square

Claim 8.22.4. $V(R_1) \cap B \subseteq V(R)$.

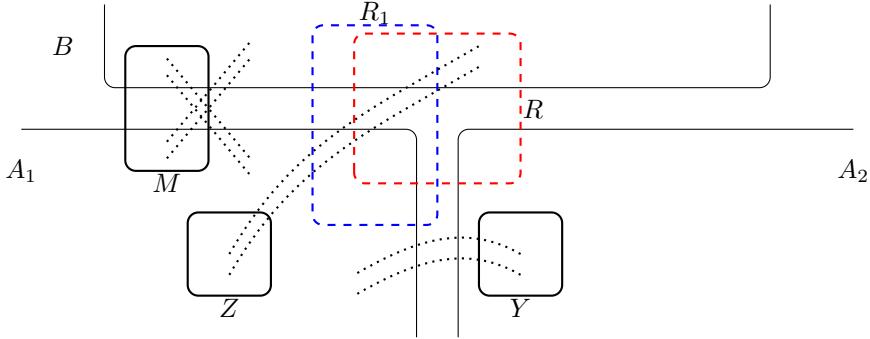
Proof. It is sufficient to prove that every vertex in $(V(F_1) \setminus V(R)) \cap B$ is not contained in R_1 . Suppose v is a vertex in $(V(F_1) \setminus V(R)) \cap B$. If v has degree at least 2 in $H_{A,B}$, then $v \in V(R)$, so we can assume that v has degree at most 1 in $H_{A,B}$. If v is isolated in F_1 , then $v \notin V(R_1)$, so v has degree 1 in F_1 . Let w be the neighbor of v in F_1 . If w has degree at least 2 in F_1 , then v is removed by definition of a reduced forest. So, vw is a single-edge component of F_1 , and since $v \notin V(R)$, we have $w \in V(R)$. Thus, by Single-edge Rule I, we have that $v \notin V(R_1)$ and $w \in V(R_1)$. We conclude that $V(R_1) \cap B \subseteq V(R)$. \square

Claim 8.22.5. Every vertex in $(V(R_1) \setminus V(R)) \cap A_1$ has at most one neighbor in $V(R) \cap B$.

Proof. Suppose not and let $v \in (V(R_1) \setminus V(R)) \cap A_1$ such that v has two neighbors x and y in $V(R) \cap B$. Clearly, $\{v, x, y\} \subseteq V(H)$. But then, $v \in V(R)$ by the definition of reduced forests, a contradiction. \square

We now construct a minimal vertex cover M_1 of $G_{A_1, A_2 \cup B} - V(R_1)$ which avoids F_1 , and verify the third condition of Definition 8.20. See Figure 8.6 for an illustration of Y and Z that are constructed below.

Note that there may be an edge between $(V(R) \setminus V(R_1)) \cap B$ and $A_1 \setminus V(F_1) \setminus (M \cap A_1)$, which is not hit by M . For example, it is possible that a vertex $a \in A_1 \setminus V(F_1) \setminus (M \cap A_1)$ and a vertex $b \in (V(R) \setminus V(R_1)) \cap B$ are adjacent in G (but not in H) and a was a potential leaf of b with respect to R and M , but b has only neighbors on A_2 in $H_{A,B}$, so that $b \in V(R)$. In this case, when we look at

Figure 8.6: An illustration of Y and Z .

$G_{A_1, A_2 \cup B} - V(R_1)$, a and b are not contained in $V(R_1)$ and a is not contained in $M \cap A_1$. To hit such edges, we define Z as the set of all vertices in $A_1 \setminus V(F_1) \setminus (M \cap A_1)$ having a neighbor in $(V(R) \setminus V(R_1)) \cap B$.

We also need to hit all edges between A_1 and A_2 in $G_{A_1, A_2 \cup B} - V(R_1)$. We use vertices in A_2 to hit these edges. We define Y to be the set of all vertices in $A_2 \setminus V(F_1)$ having a neighbor in $A_1 \setminus V(R_1) \setminus (M \cap A_1)$.

Let $M' := M \cup Y \cup Z$. We first show that M' is a vertex cover of $G_{A_1, A_2 \cup B} - V(R_1)$.

Claim 8.22.6. *The set M' is a vertex cover of $G_{A_1, A_2 \cup B} - V(R_1)$.*

Proof. Suppose there is an edge yz in $G_{A_1, A_2 \cup B} - V(R_1)$ not covered by M' . As Y hits all edges between A_1 and A_2 in $G_{A_1, A_2 \cup B} - V(R_1)$, this edge is an edge between A_1 and B . Assume that $y \in A_1$ and $z \in B$.

As $V(R) \cap A_1 \subseteq V(R_1)$, z cannot be in $B \setminus (V(R) \cup M)$, and thus, $z \in (V(R) \setminus V(R_1)) \cap B$. However, since Z covers all edges between $A_1 \setminus V(R_1) \setminus (M \cap A_1)$ and $(V(R) \setminus V(R_1)) \cap B$, y should be contained in Z , a contradiction. Therefore, M' is a vertex cover of $G_{A_1, A_2 \cup B} - V(R_1)$. \square

Now, we take a minimal vertex cover M_1 of $G_{A_1, A_2 \cup B} - V(R_1)$ contained in M' . Clearly, the set M_1 is a minimal vertex cover of $G_{A_1, A_2 \cup B} - V(R_1)$ satisfying that $M \cap A_1 \subseteq M_1$ and $M_1 \cap B \subseteq M$ by construction. So, M_1 satisfies the third condition of Definition 8.20 and (R_1, M_1) is a restriction of (R, M) .

It remains to show that F_1^* respects (R_1, M_1) . By construction, R_1 is a reduced forest of F_1 so we only have to show that $V(F_1^*) \cap M_1 = \emptyset$, and in particular, by the construction, it suffices to prove that $Z \cap V(F_1^*) = \emptyset$.

Claim 8.22.7. $Z \cap V(F_1^*) = \emptyset$.

Proof. Suppose not; let $x \in Z \cap V(F_1^*)$. Because $x \notin V(F_1)$, x has no neighbor in B in $G[A_1 \cup \text{bd}(B)]$. Therefore, $x \notin Z$, by definition. \square

We conclude that F_1^* respects (R_1, M_1) . \square

Proposition 8.23. *Let (A_1, A_2, B) be a vertex partition of a graph G . Let R be an induced forest in $G_{A_1 \cup A_2, B}$ and M be a minimal vertex cover of $G_{A_1 \cup A_2, B} - V(R)$. Let H be an induced forest in $G[A_1 \cup A_2 \cup \text{bd}(B)] - E(G[\text{bd}(B)])$ respecting (R, M) and for each $i \in \{1, 2\}$,*

- let (R_i, M_i) be a restriction of (R, M) such that

$$H \cap G[A_i \cup \text{bd}(A_{3-i} \cup B)] - E(G[\text{bd}(A_{3-i} \cup B)])$$

respects (guaranteed by Proposition 8.22), and

- let P_i be the partition of $\mathcal{C}(R_i)$ such that for $C, C' \in \mathcal{C}(R_i)$, C and C' are in the same part if and only if they are contained in the same connected component of $H \cap G[A_i \cup \text{bd}(A_{3-i} \cup B)] - E(G[\text{bd}(A_{3-i} \cup B)])$.

Then (R, R_1, R_2, P_1, P_2) is compatible.

Proof. Let Q be the auxiliary graph of (R, R_1, R_2, P_1, P_2) . It is not difficult to see that if Q contains a cycle, then H also contains a cycle, which leads to a contradiction. Thus, Q has no cycles. \square

Bottom to Top

Proposition 8.24. *Let (A_1, A_2, B) be a vertex partition of a graph G . Let R be an induced forest in $G_{A_1 \cup A_2, B}$ and M be a minimal vertex cover of $G_{A_1 \cup A_2, B} - V(R)$ such that for every vertex x of degree at most 1 in R , $PL_{R, M}(x) \neq \emptyset$. For each $i \in \{1, 2\}$,*

- let R_i be an induced forest in $G_{A_i, A_{3-i} \cup B}$ and M_i be a minimal vertex cover of $G_{A_i, A_{3-i} \cup B} - V(R_i)$, and H_i be an induced forest in $G[A_i \cup \text{bd}(A_{3-i} \cup B)] - E(G[\text{bd}(A_{3-i} \cup B)])$ respecting (R_i, M_i) ,
- let P_i be the partition of $\mathcal{C}(R_i)$ such that for $C, C' \in \mathcal{C}(R_i)$, C and C' are in the same part if and only if they are contained in the same connected component of H_i ,
- (R_i, M_i) is a restriction of (R, M) .

Furthermore,

- for each $i \in \{1, 2\}$, $V(R_i) \cap A_{3-i} \subseteq V(R_{3-i})$,
- every vertex in $(V(R) \setminus (V(R_1) \cup V(R_2))) \cap B$ has at least two neighbors in $(V(R_1) \cap A_1) \cup (V(R_2) \cap A_2)$,
- (R, R_1, R_2, P_1, P_2) is compatible.

Then there is an induced forest H in $G[A_1 \cup A_2 \cup \text{bd}(B)] - E(G[\text{bd}(B)])$ respecting (R, M) such that

$$\cdot V(H) \cap (A_1 \cup A_2) = (V(H_1) \cap A_1) \cup (V(H_2) \cap A_2).$$

Proof. For each $i \in \{1, 2\}$, we obtain H'_i from H_i by removing all vertices that are contained in $(A_{3-i} \cup B) \setminus V(R_i)$. This procedure achieves that $V(H'_i) \cap V(G_{A_i, A_{3-i} \cup B}) = V(R_i) \cap V(G_{A_i, A_{3-i} \cup B})$. We take a new graph

$$H^* := G[V(H'_1) \cup V(H'_2) \cup V(R)].$$

As (R, R_1, R_2, P_1, P_2) is compatible, we can verify that H^* is a forest. Let H be the graph obtained from $H^* - (B \setminus V(R))$ by adding a potential leaf to each vertex in $V(R) \cap (A_1 \cup A_2)$ of degree at most 1 in R and then removing newly created edges between vertices contained in B . We show that H is a forest.

Claim 8.24.1. H is a forest such that $V(H) \cap (A_1 \cup A_2) = (V(H_1) \cap A_1) \cup (V(H_2) \cap A_2)$.

Proof. Since H^* is a forest, $H^* - (B \setminus V(R))$ is also a forest. Adding a potential leaf of a vertex in $V(R) \cap (A_1 \cup A_2)$ preserves the property of being a forest, as we removed all edges in B . When we take H from H^* , we only change the vertices in B . Also, for each $i \in \{1, 2\}$, we have that

- $V(R) \cap A_i \subseteq V(R_i) \subseteq V(H_i)$ by the first condition of Definition 8.20, and
- $V(R_i) \cap A_{3-i} \subseteq V(R_{3-i})$ by the precondition of this proposition.

Therefore, we have $V(H) \cap (A_1 \cup A_2) = (V(H_1) \cap A_1) \cup (V(H_2) \cap A_2)$. □

In the remainder, we prove that H respects (R, M) . We need to verify that

- (i) R is a reduced forest of $G_{A_1 \cup A_2, B} \cap H$.
- (ii) $V(H) \cap M = \emptyset$.

Condition (ii) is easy to verify. Since we remove all vertices in $M \cap B \subseteq B \setminus V(R)$ when we construct H from H^* and then add only potential leaves with respect to R and M , we have $V(H) \cap (M \cap B) = \emptyset$. Furthermore, $V(H) \cap (M \cap (A_1 \cup A_2)) = \emptyset$ because

- $V(H) \cap (A_1 \cup A_2) = (V(H_1) \cap A_1) \cup (V(H_2) \cap A_2)$,
- for each $i \in \{1, 2\}$, $M \cap A_i \subseteq M_i$ by the third condition of Definition 8.20.

We now verify condition (i). Let $H_{A,B} := H \cap G_{A_1 \cup A_2, B}$. We first verify the following.

Claim 8.24.2. Every vertex of $V(H_{A,B}) \setminus V(R)$ has degree at most 1 in $H_{A,B}$.

Proof. Let $v \in V(H_{A,B}) \setminus V(R)$. First assume that $v \in A_1 \cup A_2$. Without loss of generality, we assume that $v \in A_1$. Since M is a vertex cover of $G_{A_1 \cup A_2, B} - V(R)$, the neighborhood of v in $H_{A,B}$ is contained in $V(R) \cap B$.

Suppose for contradiction that in $H_{A,B}$, v has at least two neighbors in $V(R) \cap B$. Since (R_1, M_1) is a restriction of (R, M) , by the second condition of Definition 8.20, v is not contained in R_1 . If v has at least two neighbors in $V(R_1) \cap B$, then v should be contained in R_1 , a contradiction. Therefore, v has at least one neighbor in $(V(R) \setminus V(R_1)) \cap B$, say w . Then vw is an edge of $H_1 \cap G_{A_1, A_2 \cup B} - V(R_1)$, which contradicts the assumption that R_1 is a reduced forest of $H_1 \cap G_{A_1, A_2 \cup B}$. Therefore, v has at most one neighbor in $V(R) \cap B$, as required.

Now we assume $v \in B$. By construction, v is a potential leaf of some vertex in R . Thus v has degree 1 in $H_{A,B}$, as required. \square

We argue that R is a reduced forest of $H_{A,B}$. Let $v \in V(R)$. If v has degree at least 2 in $H_{A,B}$, then v is contained in any reduced forest of $H_{A,B}$. Suppose v has degree at most 1 in $H_{A,B}$.

Suppose $v \in A_1 \cup A_2$. In this case, by construction, v is incident with its potential leaf in $H_{A,B}$, say w . This means that vw is a single-edge component in $H_{A,B}$, and we can take v as a vertex in R .

Now, suppose $v \in B$. First assume that $v \in V(R_i)$ for some $i \in \{1, 2\}$. If v has degree 1 in R_i , then it also has at least one potential leaf in $H_i \cap G_{A_i, A_{3-i} \cup B}$, and thus v has degree 2 in $H_{A,B}$, a contradiction. Thus, v has no neighbor in R_i , and has exactly one potential leaf, say w . By Claim 8.24.2, v is the unique neighbor of w in R , and thus vw is a single-edge component of $H_{A,B}$. Thus, we can take v as a vertex in R . Suppose $v \in (V(R) \setminus (V(R_1) \cup V(R_2))) \cap B$. Then by the precondition, it has at least two neighbors in $(V(R_1) \cap A_1) \cup (V(R_2) \cap A_2) \subseteq (V(H_1) \cap A_1) \cup (V(H_2) \cap A_2)$. Therefore, it is contained in any reduced forest of $H_{A,B}$. With Claim 8.24.1, it shows that R is a reduced forest of $H_{A,B}$. \square

8.3.2 The Algorithm

In this section we give an algorithm that solves the FEEDBACK VERTEX SET problem on graphs on n vertices together with one of its branch decomposition of mim-width w in time $n^{\mathcal{O}(w)}$. We first give an algorithm for the unweighted version of the problem and then argue how to modify it for the weighted version.

First, we observe that given a graph G , a subset of its vertex set $S \subseteq V(G)$ is by definition a feedback vertex set if and only if $G - S$ is an induced forest; that is, $V(G) \setminus S$ induces a forest. It is therefore readily seen that computing the minimum size of a feedback vertex set is equivalent to computing the maximum size of an induced forest, so in the remainder of this section we solve the following problem which is more convenient for our exposition.

MAXIMUM INDUCED FOREST parameterized by $w := \text{mimw}(T, \mathcal{L})$.

Input: A graph G on n vertices, a branch decomposition (T, \mathcal{L}) of G and an integer k .

Question: Does G contain an induced forest having at least k vertices?

We furthermore assume that G is connected; otherwise, we can solve it for each connected component. Also, we assume that G contains at least 2 vertices.

We solve the MAXIMUM INDUCED FOREST problem by bottom-up dynamic programming over (T, \mathcal{L}) , the given branch decomposition of G , starting at the leaves of T . Let $t \in V(T)$ be a node of T . To motivate the table indices of the dynamic programming table, we now observe how a solution to MAXIMUM INDUCED FOREST, an induced forest \mathcal{F} , interacts with the graph $G_{t+\text{bd}} := G[V_t \cup \text{bd}(\bar{V}_t)] - E(G[\text{bd}(\bar{V}_t)])$. The intersection of \mathcal{F} with $G_{t+\text{bd}}$ is an induced forest which throughout the following we denote by $\mathcal{F}_{t+\text{bd}} := \mathcal{F} \cap G_{t+\text{bd}}$. Since we want to bound the number of table entries by $n^{\mathcal{O}(w)}$, we have to focus in particular on the interaction of \mathcal{F} with the crossing graph $G_{t,\bar{t}}$ which is an induced forest in $G_{t,\bar{t}}$, denoted by $\mathcal{F}_{t,\bar{t}} := \mathcal{F}[V(G_{t,\bar{t}})]$.

However, it is not possible to enumerate all induced forests in a crossing graph as potential table indices: Consider for example a star on n vertices and the cut consisting of the central vertex on one side and the remaining vertices on the other side. This cut has mim-value 1 but it contains 2^n induced forests, since each vertex subset of the star induces a forest on the cut. The remedy for this issue are *reduced* (induced) forests, introduced in Section 8.3.1.

In order to avoid having exponentially (in n) many table entries at each node $t \in V(T)$, we use all reduced forests of $G_{t,\bar{t}}$ to represent the ways in which induced forests can intersect with $G_{t,\bar{t}}$ as parts of the table entries. By Lemma 8.16, the number of reduced forests in each cut of mim-value w is bounded by $\mathcal{O}(n^{6w})$. However, reduced forests alone do not provide enough information about induced forests in $G_{t,\bar{t}}$. We now analyze the structure of $\mathcal{F}_{t,\bar{t}}$ to motivate the additional objects that can be used to represent $\mathcal{F}_{t,\bar{t}}$ in such a way that the number of all possible table entries remains bounded by $n^{\mathcal{O}(w)}$.

Let \mathfrak{R} be a reduced forest of $\mathcal{F}_{t,\bar{t}}$. The induced forest $\mathcal{F}_{t,\bar{t}}$ has three types of vertices in $G_{t,\bar{t}}$:

- The vertices of the reduced forest \mathfrak{R} .
- The leaves of the induced forest $\mathcal{F}_{t,\bar{t}}$ that are not contained in \mathfrak{R} , denoted by $L(\mathcal{F}_{t,\bar{t}})$.
- Vertices in $\mathcal{F}_{t,\bar{t}}$ that do not have a neighbor in $\mathcal{F}_{t,\bar{t}}$ on the opposite side of the boundary, in the following called *non-crossing* vertices and denoted by $NC(\mathcal{F}_{t,\bar{t}})$.

As outlined above, the only type of vertices in $\mathcal{F}_{t,\bar{t}}$ that will be used as part of the table indices are the vertices of a reduced forest of $\mathcal{F}_{t,\bar{t}}$, since otherwise, the

number of possible indices might be exponential in n . Hence, we neither know about the leaves of $\mathcal{F}_{t,\bar{t}}$ (apart from components that are single edges) nor its non-crossing vertices upon inspecting this part of the index. Suppose we have a vertex $v \in (\text{L}(\mathcal{F}_{t,\bar{t}}) \cup \text{NC}(\mathcal{F}_{t,\bar{t}})) \cap V_t$ and consider $N_t^*(v) := (N_G(v) \cap \bar{V}_t) \setminus V(\mathfrak{R})$. Then, $\mathcal{F}_{t,\bar{t}}$ does not use any vertex x in $N_t^*(v)$: If v is a leaf in $\mathcal{F}_{t,\bar{t}}$, then the presence of the edge $\{v, x\}$ would make it a non-leaf vertex and if v is a non-crossing vertex, the presence of $\{v, x\}$ would make v a vertex incident with an edge of the forest crossing the cut. An analogous point can be made for a vertex in $(\text{L}(\mathcal{F}_{t,\bar{t}}) \cup \text{NC}(\mathcal{F}_{t,\bar{t}})) \cap \bar{V}_t$. In the table indices, we capture this property of $\mathcal{F}_{t,\bar{t}}$ by considering a minimal vertex cover of $G_{t,\bar{t}} - V(\mathfrak{R})$ that avoids all leaves and non-crossing vertices of $\mathcal{F}_{t,\bar{t}}$. We observe that such a minimal vertex cover always exists. (Note that $\text{L}(\mathcal{F}_{t,\bar{t}}) \cup \text{NC}(\mathcal{F}_{t,\bar{t}})$ is an independent set in $G_{t,\bar{t}}$.)

Observation 8.25. Let G be a graph and $X \subseteq V(G)$ an independent set in G . Then, there exists a minimal vertex cover M of G such that $X \cap M = \emptyset$.

Lastly, we have to keep track of how the connected components of $\mathcal{F}_{t,\bar{t}}$ (respectively, \mathfrak{R}) are joined together via the forest $\mathcal{F}_{t+\text{bd}}$. This forest induces a partition of $\mathcal{C}(\mathfrak{R})$ in the following way: Two components $C_1, C_2 \in \mathcal{C}(\mathfrak{R})$ are in the same part of the partition if and only if C_1 and C_2 are contained in the same connected component of $\mathcal{F}_{t+\text{bd}}$.

We are now ready to define the indices of the dynamic programming table **tab** to keep track of sufficiently much information about the partial solutions in the graph $G_{t+\text{bd}}$. Throughout the following, we denote by \mathcal{R}_t the set of all induced forests of $G_{t,\bar{t}}$ on at most $6w$ vertices (which by Lemma 8.16 contains all reduced forests in $G_{t,\bar{t}}$). For $R \in \mathcal{R}_t$, we let $\mathcal{M}_{t,R}$ be the set of all minimal vertex covers of $G_{t,\bar{t}} - V(R)$ and $\mathcal{P}_{t,R}$ the set of all partitions of the connected components of R .

For an illustration of the above discussion and also the definition of the table indices, which we start on now, see Figure 8.7. For $(R, M, P) \in \mathcal{R}_t \times \mathcal{M}_{t,R} \times \mathcal{P}_{t,R}$ and $i \in \{0, \dots, n\}$, we set $\text{tab}[t, (R, M, P), i] := 1$ (and to 0 otherwise), if and only if the following conditions are satisfied.

- (i) There is an induced forest F in $G[V_t \cup \text{bd}(\bar{V}_t)] - E(G[\text{bd}(\bar{V}_t)])$, such that $V(F) \cap V_t$ has size i .
- (ii) Let $F_{t,\bar{t}} = F \cap G_{t,\bar{t}}$, i.e. $F_{t,\bar{t}}$ is the subforest of F induced by the vertices of the crossing graph $G_{t,\bar{t}}$. Then, R is a reduced forest of $F_{t,\bar{t}}$.
- (iii) M is a minimal vertex cover of $G_{t,\bar{t}} - V(R)$ such that $V(F) \cap M = \emptyset$.
- (iv) P is a partition of $\mathcal{C}(R)$ such that two components $C_1, C_2 \in \mathcal{C}(R)$ are in the same part of the partition if and only if C_1 and C_2 are contained in the same connected component of F .

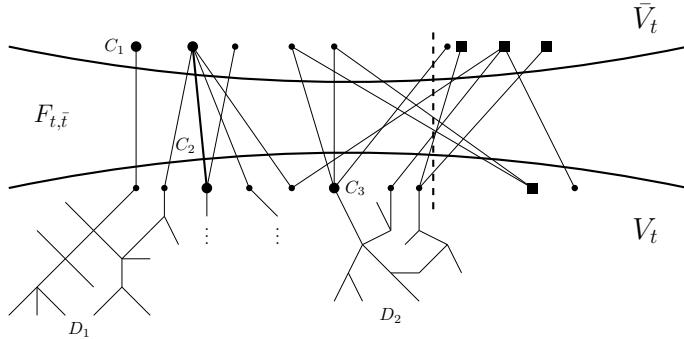


Figure 8.7: An example of a crossing graph $G_{t,\bar{t}}$ together with an induced forest \mathcal{F} and their interaction. The forest $\mathcal{F}_{t,\bar{t}} = \mathcal{F}[V(G_{t,\bar{t}})]$ is displayed to the left of the dividing line in the drawing and the 4 vertices and 1 edge in bold form a reduced forest R of $\mathcal{F}_{t,\bar{t}}$. The square vertices form a minimal vertex cover of $G_{t,\bar{t}} - V(R)$ satisfying (iii). Furthermore, C_i ($i \in [3]$) are the connected components of R and D_i ($i \in [2]$) are the connected components of \mathcal{F} .

For a node $t \in V(T)$, we let tab_t be the subtable of tab with respect to t as the table entries that have t as a first index. I.e. for $(R, M, P) \in \mathcal{R}_t \times \mathcal{M}_{t,R} \times \mathcal{P}_{t,R}$ and $i \in \{0, \dots, n\}$, we let $\text{tab}_t[(R, M, P), i] := \text{tab}[t, (R, M, P), i]$. Note that (ii) and (iii) express that F respects (R, M) . Regarding (iii), recall that even though the leaves and non-crossing vertices of $F_{t,\bar{t}}$ are still contained in $G_{t,\bar{t}} - V(R)$, a minimal vertex cover that avoids the leaves and non-crossing vertices of $F_{t,\bar{t}}$ always exists by Observation 8.25.

Recall that $r \in V(T)$ denotes the root of T , the tree of the given branch decomposition of G . From Property (i) we immediately observe that the table entries store enough information to obtain a solution to MAXIMUM INDUCED FOREST after all table entries have been filled. In particular, we make

Observation 8.26. The graph G contains an induced forest on i vertices if and only if $\text{tab}[r, (\emptyset, \emptyset, \emptyset), i] = 1$.

Before we proceed with the description of the algorithm, we first show that the number of table entries is bounded by a polynomial whose degree is linear in the mim-width w of the given branch decomposition.

Proposition 8.27. *There are at most $n^{\mathcal{O}(w)}$ table entries in tab .*

Proof. Let $t \in V(T)$. We show that the number of table entries in tab_t is bounded by $n^{\mathcal{O}(w)}$ which together with the observation that $|V(T)| = \mathcal{O}(n)$ yields the proposition. By definition, $|\mathcal{R}_t| = \mathcal{O}(n^{6w})$ and by the Minimal Vertex Covers Lemma we have for each $R \in \mathcal{R}_t$ that $|\mathcal{M}_{t,R}| = n^{\mathcal{O}(w)}$. The size of $\mathcal{P}_{t,R}$ is at most the number of

partitions of a set of size $6w$, and hence at most $B_{6w} < (w/\log(w))^{\mathcal{O}(w)}$ by standard upper bounds on the Bell number B_{6w} . Finally, there are $n+1$ choices for the integer i . To summarize, there are at most

$$\mathcal{O}(n^{6w}) \cdot n^{\mathcal{O}(w)} \cdot (w/\log(w))^{\mathcal{O}(w)} \cdot (n+1) = n^{\mathcal{O}(w)}$$

table entries in tab_t and the proposition follows. \square

We now show how to compute the table entries in tab . First, we explain how to compute the entries in tab_ℓ for the leaves ℓ of T and then how to compute the entries in the internal nodes of T from the entries stored in the tables corresponding to their children.

Leaves of T . Let $t \in V(T)$ be a leaf of T and $v = \mathcal{L}^{-1}(t)$. Clearly, the crossing graph $G_{t,\bar{t}}$ is a star S with central vertex v or a single edge. Hence, any induced forest F in $G[\{v\} \cup N(v)] - E(G[N(v)])$ satisfies that either $V(F) = \{v\}$ or $V(F) \subseteq N(v)$ or F contains an edge in $G_{t,\bar{t}}$. In the last case, either F is a single edge or a star with central vertex v . Let R be a reduced forest of F . The cases we have to consider to fill the table entries are the following.

If $F = \emptyset$, then both $\{v\}$ and $N(v)$ are feasible minimal vertex covers and clearly, $P = \emptyset$. If $V(F) = \{v\}$, then $R = \emptyset$, $M = N(v)$, $P = \emptyset$, and $i = 1$. If $V(F) \subseteq N(v)$, then $R = \emptyset$, $M = \{v\}$, $P = \emptyset$, and $i = 0$. Throughout the following, we assume that F contains an edge in $G_{t,\bar{t}}$.

Suppose F is a single edge $\{v, w\}$. Then, R is either the vertex v or the vertex w . If $V(R) = \{v\}$, then $G_{t,\bar{t}} - V(R)$ does not contain any edges and hence $\mathcal{M}_{t,R} = \{\emptyset\}$. Furthermore, F has size one in $G[V_t] = G[\{v\}]$. If $V(R) = \{w\}$, then v is a leaf in F and hence the only minimal vertex cover satisfying (iii) is the set of neighbors of v without w , i.e. the set $N(v) \setminus \{w\}$. The size of F in $G[V_t]$ is 1. In both cases, F only has one component, so $\mathcal{P}_{t,R} = \{\{R\}\}$.

Now suppose that F has at least three vertices. Then, F is a star with central vertex v and hence, the reduced forest of any such F is the single vertex v . Since the vertices of F in $\overline{V_t}$ are not counted in the table entry by (i), we only have to consider one index where the reduced forest is v , the minimal vertex cover is empty (again since $G_{t,\bar{t}} - \{v\}$ does not have any edges), the partition of R is the singleton partition and $i = 1$, since F has size one in $G[V_t] = G[\{v\}]$. To summarize, the table entries for the leaf t are set as follows. We let:

$$\begin{aligned} \text{tab}[t, (R, M, P), i] := \\ \left\{ \begin{array}{ll} 1, & \text{if } R = \emptyset, M \in \{\{v\}, N(v)\}, P = \emptyset, i = 0 \\ 1, & \text{if } R = \emptyset, M = N(v), P = \emptyset, i = 1 \\ 1, & \text{if } R = G[\{v\}], M = \emptyset, P = \{R\}, i = 1 \\ 1, & \text{if } R = G[\{w\}] \text{ where } w \in N(v), M = N(v) \setminus \{w\}, \\ & P = \{R\}, i = 1 \\ 0, & \text{otherwise} \end{array} \right. \end{aligned}$$

Internal Nodes of T . Let $t \in V(T)$ be an internal node with children a and b . Using Propositions 8.22, 8.23 and 8.24, we can show the following.

Proposition 8.28. *Let $\mathfrak{I} = [(R, M, P), i] \in (\mathcal{R}_t \times \mathcal{M}_{t, R_t} \times \mathcal{P}_{t, R_t}) \times \{0, \dots, n\}$ such that for every vertex x of degree at most 1 in R , $PL_{R, M}(x) \neq \emptyset$. Then $\text{tab}[t, (R, M, P), i] = 1$ if and only if there are restrictions (R_a, M_a) and (R_b, M_b) of (R, M) to $G_{a, \bar{a}}$ and $G_{b, \bar{b}}$, respectively, and partitions P_a and P_b of $\mathcal{C}(R_a)$ and $\mathcal{C}(R_b)$, respectively, and integers i_a and i_b such that*

- $\text{tab}[t_a, (R_a, M_a, P_a), i_a] = 1$ and $\text{tab}[t_b, (R_b, M_b, P_b), i_b] = 1$,
- (R, R_a, R_b, P_a, P_b) is compatible and $P = \mathcal{U}(R, R_a, R_b, P_a, P_b)$,
- every vertex in $(V(R) \setminus (V(R_a) \cup V(R_b))) \cap B$ has at least two neighbors in $(V(R_a) \cap V_a) \cup (V(R_b) \cap V_b)$,
- $V(R_a) \cap V_b \subseteq V(R_b)$ and $V(R_b) \cap V_a \subseteq V(R_a)$,
- $i_a + i_b = i$.

Proof. Suppose $\text{tab}[t, (R, M, P), i] = 1$. Let H be an induced forest of $G[V_t \cup \text{bd}(\overline{V_t})] - E(G[\text{bd}(\overline{V_t})])$ that is a partial solution with respect to (R, M, P) and i . For each $x \in \{a, b\}$, let $H_x := H \cap (G[V_x \cup \text{bd}(\overline{V_x})] - E(G[\text{bd}(\overline{V_x})]))$. By Proposition 8.22, there are restrictions (R_a, M_a) and (R_b, M_b) of (R, M) to V_a and V_b , respectively, such that

- H_a respects (R_a, M_a) , and H_b respects (R_b, M_b) , and
- every vertex in $(V(R) \setminus (V(R_a) \cup V(R_b))) \cap B$ has at least two neighbors in $(V(R_a) \cap V_a) \cup (V(R_b) \cap V_b)$,
- $V(R_a) \cap V_b \subseteq V(R_b)$ and $V(R_b) \cap V_a \subseteq V(R_a)$.

For each $x \in \{a, b\}$, let P_x be the partition of $\mathcal{C}(R_x)$ such that two graphs in $\mathcal{C}(R_x)$ are contained in the same part if and only if they are contained in the same connected component of H_x . Then by Proposition 8.23, the tuple (R, R_a, R_b, P_a, P_b) is compatible and it is not difficult to verify that $P = \mathcal{U}(R, R_a, R_b, P_a, P_b)$. Let $i_x := |V(H) \cap V(G[V_x])|$. Then, $i_a + i_b = i$ as V_a and V_b are disjoint. This concludes the forward direction.

To verify the converse direction, suppose the latter conditions hold. For each $x \in \{a, b\}$, let H_x be an induced forest in $G[V_x \cup \text{bd}(\overline{V_x})] - E(G[\text{bd}(\overline{V_x})])$ that is a partial solution with respect to (R_x, M_x, P_x) and i_x . By the second, third, and fourth condition, we can apply Proposition 8.24 to conclude that there is an induced forest H in $G[V_t \cup \text{bd}(\overline{V_t})] - E(G[\text{bd}(\overline{V_t})])$ respecting (R, M) such that

$$H \cap G[V_t] = (H_a \cap G[V_a]) \cup (H_b \cap G[V_b]).$$

Therefore, we have $|V(H) \cap V_t| = |V(H_a) \cap V_a| + |V(H_b) \cap V_b| = i_a + i_b = i$, so $\text{tab}[t, (R, M, P), i] = 1$, as required. \square

Based on Proposition 8.28, we can proceed with the computation of the table at an internal node t with children a and b . Let $\mathfrak{I} = [(R, M, P), i] \in (\mathcal{R}_t \times \mathcal{M}_{t,R_t} \times \mathcal{P}_{t,R_t}) \times \{0, \dots, n\}$.

Step 1 (Valid Index). We verify whether \mathfrak{I} is valid, i.e. whether it can represent a valid partial solution in the sense of the definition of the table entries. That is, each vertex of degree at most 1 in R has to have at least one potential leaf.

Step 2 (Reduced Forests). We consider all pairs of indices for tab_a and tab_b denoted by

- $\mathfrak{I}_a = [(R_a, M_a, P_a), i_a] \in (\mathcal{R}_a \times \mathcal{M}_{a,R_a} \times \mathcal{P}_{a,R_a}) \times \{0, \dots, n\}$ and
- $\mathfrak{I}_b = [(R_b, M_b, P_b), i_b] \in (\mathcal{R}_b \times \mathcal{M}_{b,R_b} \times \mathcal{P}_{b,R_b}) \times \{0, \dots, n\}$.

We check

- (R_a, M_a) and (R_b, M_b) are restrictions of (R, M) to $G_{a,\bar{a}}$ and $G_{b,\bar{b}}$ respectively,
- $\text{tab}[t_a, (R_a, M_a, P_a), i_a] = 1$ and $\text{tab}[t_b, (R_b, M_b, P_b), i_b] = 1$,
- (R, R_a, R_b, P_a, P_b) is compatible and $P = \mathcal{U}(R, R_a, R_b, P_a, P_b)$,
- every vertex in $(V(R) \setminus (V(R_a) \cup V(R_b))) \cap B$ has at least two neighbors in $(V(R_a) \cap V_a) \cup (V(R_b) \cap V_b)$,
- $V(R_a) \cap V_b \subseteq V(R_b)$ and $V(R_b) \cap V_a \subseteq V(R_a)$,
- $i_a + i_b = i$.

If there is a pair $\mathfrak{I}_a, \mathfrak{I}_b$ of table indices satisfying all of the above conditions, then we assign $\text{tab}[t, (R, M, P), i] := 1$ and otherwise, we assign $\text{tab}[t, (R, M, P), i] := 0$. Correctness follows from Proposition 8.28.

We finish by analyzing the running time of the algorithm. At each node $t \in V(T)$, we can enumerate all table indices in time $n^{\mathcal{O}(w)}$ by Corollary 8.3 and Proposition 8.27. Let $\mathfrak{I} = [(R, M, P), i] \in (\mathcal{R}_t \times \mathcal{M}_{t,R_t} \times \mathcal{P}_{t,R_t}) \times \{0, \dots, n\}$. If t is a leaf node, then $\text{tab}[t, (R, M, P), i]$ can be computed in linear time. Assume that t is an internal node. We can check in linear time whether \mathfrak{I} is valid or not. Next, for all pairs of $\mathfrak{I}_a = [(R_a, M_a, P_a), i_a] \in (\mathcal{R}_a \times \mathcal{M}_{a,R_a} \times \mathcal{P}_{a,R_a}) \times \{0, \dots, n\}$ and $\mathfrak{I}_b = [(R_b, M_b, P_b), i_b] \in (\mathcal{R}_b \times \mathcal{M}_{b,R_b} \times \mathcal{P}_{b,R_b}) \times \{0, \dots, n\}$ we verify that the conditions of Step 2 hold, which can be done in time $\mathcal{O}(n^2)$. Therefore, by Proposition 8.27, we can decide whether $\text{tab}[t, (R, M, P), i] = 1$ or not in time $n^{\mathcal{O}(w)}$. As T contains $\mathcal{O}(n)$ nodes, we can solve MAXIMUM INDUCED FOREST, and by duality FEEDBACK VERTEX SET in time $n^{\mathcal{O}(w)}$.

We can easily modify our algorithm into an algorithm solving the weighted version of the problem. In WEIGHTED FEEDBACK VERTEX SET, we are given a graph and a

weight function $\omega : V(G) \rightarrow \mathbb{R}$, we want to find a set S with minimum $\omega(S)$ such that $G - S$ has no cycles. Similar to FEEDBACK VERTEX SET, we can instead solve the problem of finding an induced forest F with maximum $\omega(V(F))$. Instead of specifying i in the table index $[t, (R, M, P), i]$, we store at $\text{tab}[t, (R, M, P)]$ the maximum value $\omega(V(F) \cap V_t)$ over all induced forests F that respect (R, M) and whose connectivity partition is P . The procedure for leaf nodes is analogous. In the internal node, we compare all pairs (R_a, M_a, P_a) and (R_b, M_b, P_b) for children t_a and t_b , and take the maximum among all sums $\mathcal{T}[t_a, (R_a, M_a, P_a)] + \mathcal{T}[t_b, (R_b, M_b, P_b)]$. Therefore, we can solve WEIGHTED FEEDBACK VERTEX SET (and MAXIMUM WEIGHT INDUCED FOREST) in time $n^{\mathcal{O}(w)}$ as well. We have proved Theorem 8.14.

Our algorithm can furthermore be used to solve the connected variant of the MAXIMUM (WEIGHT) INDUCED FOREST problem, namely MAXIMUM (WEIGHT) INDUCED TREE. To see this, note that one part of the table indices is the connectivity partition of all forests that correspond to a given index. Each part of this partition represents a connected component of a corresponding forest. Hence, we can solve MAXIMUM (WEIGHT) INDUCED TREE as follows. First, we compute all the table entries as when solving MAXIMUM (WEIGHT) INDUCED FOREST. Then, when reading off the solution value to the problem in the table entries corresponding to the root of the branch decomposition, we simply restrict our search to table indices whose connectivity partitions consist of a single part: these entries are precisely the ones that correspond to solutions that form a tree.

Corollary 8.29. *Given an n -vertex graph and one of its branch decompositions of mim-width w , we can solve MAXIMUM (WEIGHT) INDUCED FOREST and MAXIMUM (WEIGHT) INDUCED TREE in time $n^{\mathcal{O}(w)}$.*

8.4 Some Concluding Remarks

The algorithms we presented in this chapter rely on problem specific ad-hoc arguments. Recently, Bergougnoux and Kanté [21] provided a more generic approach to connectivity and acyclicity problems on graphs of bounded mim-width, which includes LONGEST INDUCED PATH and FEEDBACK VERTEX SET, and many more.

Nevertheless, it would be interesting to see if tools such as the Minimal Vertex Covers Lemma (Corollary 8.3) or other combinatorial observations about cuts of small mim-width can be useful in the design of efficient algorithms parameterized by mim-width. Perhaps a combination with the framework of Bergougnoux and Kanté would be fruitful.

Hardness Results on Mim-Width

In this chapter we prove hardness results for several problems parameterized by mim-width. First we show that HAMILTONIAN CYCLE is NP-complete on graphs of linear mim-width 1. In some sense, this result induces a shift in our understanding on graphs of linear mim-width 1. Until now, in all graph classes that are known to have linear mim-width 1, HAMILTONIAN CYCLE was polynomial-time solvable [100, 183, 226]. Therefore, the graph class LINEAR MIM-WIDTH 1 may be much larger than previously expected.

We then move on to discuss the first W[1]-hardness results for problems parameterized by mim-width that were obtained by Fomin et al. [127]. Based on their reductions, we give further W[1]-hardness results for several (σ, ρ) -domination problems as well as for FEEDBACK VERTEX SET.

9.1 Hamiltonian Cycle on Linear Mim-Width 1

In this section we prove that the HAMILTONIAN CYCLE problem remains NP-complete on graphs of linear mim-width 1. We recall the problem definition.

HAMILTONIAN CYCLE

Input: A graph G

Question: Is there a cycle $C \subseteq G$ such that $V(C) = V(G)$?

The proof of the next theorem is based on an NP-completeness proof on rooted directed path graphs due to Panda and Pradhan [241]. As shown in Section 4.7, the (non-linear) mim-width of the class ROOTED DIRECTED PATH is at most 1, therefore hardness for mim-width 1 can be observed directly. Nevertheless, there are rooted directed path graphs of unbounded linear mim-width, namely the trees. This is why we need a separate proof for the linear case.

Theorem 9.1. HAMILTONIAN CYCLE is NP-complete on graphs of linear mim-width 1, and the hardness holds even if a linear branch decomposition of mim-width 1 is given.

Proof. Itai et al [166] showed that given a bipartite graph G with maximum degree 3, it is NP-complete to decide if it has a Hamiltonian cycle, while Panda and Pradhan [241] construct, from this graph G , a rooted directed path graph H such that H has a Hamiltonian cycle if and only if G does. Here we show that the construction of [241] can be used to also output a linear branch decomposition of mim-width 1 of H , in polynomial time, which will prove our result.

Let G be the bipartite graph on bipartition $(\{v_1, v_2, \dots, v_m\}, \{w_1, w_2, \dots, w_m\})$ that has maximum degree at most 3, and has no leaves. Let us consider the construction, given by Panda and Pradhan [241], of a new graph H from G as follows:

- (i) For each $i \in \{1, \dots, m\}$, we introduce vertices X_i, Y_i to H , and if w_i has degree 3, then we introduce a vertex Z_i additionally. We let $\mathcal{X}_{\geq i} := \bigcup_{i' > i} X_{i'}$, $\mathcal{X} := \mathcal{X}_{\geq 1}$, $\mathcal{Y}_{\geq i} := \bigcup_{i' \geq i} Y_{i'}$, $\mathcal{Y} := \mathcal{Y}_{\geq 1}$, $\mathcal{Z}_{\geq i} := \bigcup_{\substack{i' \geq i \\ \deg(w_i)=3}} Z_{i'}$ and $\mathcal{Z} := \mathcal{Z}_{\geq 1}$.
- (ii) For each $v_i w_j \in E(G)$, we introduce a vertex $A_{i,j}$ to H . We let $\mathcal{A}_j := \bigcup_{i: v_i w_j \in E(G)} A_{i,j}$ and $\mathcal{A} := \bigcup_{j \leq m} \mathcal{A}_j$.
- (iii) For each $i \in \{1, \dots, m\}$, we make $\{X_i\} \cup \{A_{i',j} : i' \leq i, v_i w_j \in E(G)\}$ a clique in H .
- (iv) For each $j \in \{1, \dots, m\}$, we make $\{Y_j\} \cup \{A_{i,j} : v_i w_j \in E(G)\}$ a clique in H , and w_i has degree 3, then we also make $\{Z_j\} \cup \{A_{i,j} : v_i w_j \in E(G)\}$ a clique in H .

That concludes the construction of the graph H . We now summarize the most important properties of H which will be useful later in the proof.

- (H1) $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}$ is an independent set in H .
- (H2) \mathcal{A} is a clique in H .
- (H3) For $j \in [m]$ and $v \in \{Y_j, Z_j\}$, $N(v) = \mathcal{A}_j$.
- (H4) For $i_1, i_2 \in [m]$ with $i_1 \leq i_2$, $N(X_{i_1}) \subseteq N(X_{i_2})$ and for all $j \in [m]$, $N(A_{i_1,j}) \cap \mathcal{X} \subseteq N(A_{i_2,j}) \cap \mathcal{X}$.
- (H5) For $j \in [m - 1]$, no vertex in \mathcal{A}_j has a neighbor in $\mathcal{Y}_{\geq j+1} \cup \mathcal{Z}_{\geq j+1}$.

Formally, a branch decomposition (T, \mathcal{L}) of H is linear if T consists of a path on $|V(H)| - 2$ nodes with a leaf added to each inner node of the path, with \mathcal{L} a bijection between the leaves of T and the vertices of H . For simplicity, let us say that a linear branch decomposition is a total ordering of the vertices of H . For each

$j \in \{1, \dots, m\}$, let L_j be any ordering of vertices in \mathcal{A}_j , and let U_j be the ordering (Y_j, Z_j) if w_j has degree 3, and (Y_j) otherwise. We claim that the linear branch decomposition¹

$$U_1 \oplus L_1 \oplus U_2 \oplus L_2 \oplus \cdots \oplus U_m \oplus L_m \oplus (X_m, X_{m-1}, \dots, X_1) \quad (9.1)$$

of H has mim-width at most 1. Let $v \in V(H)$, and let S_v be the union of $\{v\}$ and the set of vertices appearing before v in the ordering, and let $T_v := V(H) \setminus S_v$. We divide into three cases depending on the place where the vertex v appears in the linear branch decomposition. Suppose for a contradiction that there exist $a_1, a_2 \in S_v$, $b_1, b_2 \in T_v$ such that $a_1b_1, a_2b_2 \in E(H)$ but $a_1b_2, a_2b_1 \notin E(H)$ (which would imply that the linear branch decomposition (9.1) has mim-width at least two).

Case 1 ($v \in U_k$ for some k). By (H3), every vertex in U_t where $t < k$, has no neighbors in T_v . Let $x \in T_v$ be a neighbor of v . By (H1), $x \in \mathcal{A}$ so by (H2), x is adjacent to every vertex in $L_1 \cup L_2 \cup \cdots \cup L_k$. We can conclude that v is neither a_1 nor a_2 . We have argued that $a_1, a_2 \in L_1 \cup \cdots \cup L_{k-1}$. By (H4), either $N(a_1) \cap \mathcal{X} \subseteq N(a_2) \cap \mathcal{X}$ or $N(a_2) \cap \mathcal{X} \subseteq N(a_1) \cap \mathcal{X}$. Suppose wlog. that the former holds. Together with (H2) and (H5), we can conclude that $N(a_1) \cap T_v \subseteq N(a_2) \cap T_v$, a contradiction.

Case 2 ($v \in L_k$ for some k). Again by (H3), every vertex in U_t where $t < k$ has no neighbors in T_v . By the argument given in Case 1, it cannot happen that a_1 and a_2 are both contained in $L_1 \cup \cdots \cup L_{k-1} \cup (L_k \cap S_v)$. Hence we can (wlog.) assume that $a_1 \in U_k$, i.e. $a_1 = Y_k$ or $a_1 = Z_k$. Since Y_k and Z_k are twins by (H3), a_2 cannot be the vertex in $U_k \setminus \{a_1\}$ (if exists). We can conclude that $a_2 \in L_1 \cup \cdots \cup L_{k-1} \cup (L_k \cap S_v)$, in particular that $a_2 \in \mathcal{A}$. By (H3), $N(a_1) = \mathcal{A}_k$, so by (H2), every neighbor of a_1 is adjacent to a_2 , a contradiction.

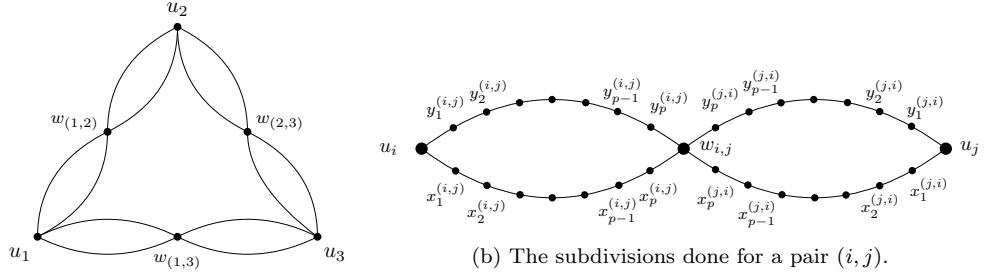
Case 3 ($v = X_k$ for some k). Suppose wlog. that $b_1 = X_{j_1}$ and $b_2 = X_{j_2}$ where $j_1 < j_2 < k$. By (H4), $N(b_1) \subset N(b_2)$, so in particular $N(b_1) \cap S_v \subset N(b_2) \cap S_v$, a contradiction.

We have shown that the linear branch decomposition (9.1) has mim-width 1. \square

9.2 Hardness Results for H -Graphs due to Fomin et al.

In this section we describe two reductions due to Fomin, Golovach and Raymond [127] that show that INDEPENDENT SET and DOMINATING SET are W[1]-hard on H -graphs parameterized by solution size plus the number of edges in H . For the definition of

¹For two disjoint total orderings X and Y we define their sum $X \oplus Y$ as follows. Suppose $X = (x_1, x_2, \dots, x_r)$ and $Y = (y_1, y_2, \dots, y_s)$, then $X \oplus Y = (x_1, x_2, \dots, x_r, y_1, y_2, \dots, y_s)$.



(a) Example of the graph H when $k = 3$

(b) The subdivisions done for a pair (i, j) .

Figure 9.1: Illustration of the construction of Fomin et al. [127].

H -graphs, see page 70. Fomin et al. [127] also showed that the linear mim-width of each H -graphs is at most $2 \cdot ||H||$, therefore the $\text{W}[1]$ -hardness carries over to the parameterization linear mim-width plus solution size. In later sections, we adapt these two reductions to prove more problems parameterized by linear mim-width $\text{W}[1]$ -hard.

9.2.1 Independent Set

We sketch the reduction due to Fomin et al. [127] that proves the following result; we refer to the original reference for several details.

Theorem 9.2 (Fomin et al. [127]). INDEPENDENT SET is $\text{W}[1]$ -hard on H -graphs parameterized by the number of edges in H plus solution size, and the hardness holds even if an H -representation of the input graph is given.

Proof (Sketch). The reduction is from MULTICOLORED CLIQUE, where given a graph G and a partition V_1, \dots, V_k of $V(G)$, the question is whether G contains a clique of size k using precisely one vertex from each V_i ($i \in [k]$). This problem is known to be $\text{W}[1]$ -complete [119, 244].

Let (G, V_1, \dots, V_k) be an instance of MULTICOLORED CLIQUE. We can assume that $k \geq 3$ and that $|V_i| = p$ for $i \in [k]$. If the second assumption does not hold, let $p := \max_{i \in [k]} |V_i|$ and add $p - |V_i|$ isolated vertices to V_i , for each $i \in [k]$. (Note that adding isolated vertices does not change the answer to the problem.) For $i \in [k]$, we denote by v_1^i, \dots, v_p^i the vertices of V_i .

The graph H is obtained as follows, see Figure 9.1a.

1. Construct k nodes u_1, \dots, u_k .
2. For every $1 \leq i < j \leq k$, construct a node $w_{i,j}$ and two pairs of parallel edges $u_i w_{i,j}$ and $u_j w_{i,j}$.

We then construct the subdivision H' of H by first subdividing each edge p times. We denote the subdivision nodes for 4 edges of H constructed for each pair $1 \leq i < j \leq k$ in Step 2. by $x_1^{(i,j)}, \dots, x_p^{(i,j)}$, $y_1^{(i,j)}, \dots, y_p^{(i,j)}$, $x_1^{(j,i)}, \dots, x_p^{(j,i)}$, and $y_1^{(j,i)}, \dots, y_p^{(j,i)}$. This subdivision process is depicted in Figure 9.1b. To simplify notation, we assume that $u_i = x_0^{(i,j)} = y_0^{(i,j)}$, $u_j = x_0^{(j,i)} = y_0^{(j,i)}$ and $w_{i,j} = x_{p+1}^{(i,j)} = y_{p+1}^{(i,j)} = x_{p+1}^{(j,i)} = y_{p+1}^{(j,i)}$.

We now construct G' by defining its H -representation $\mathcal{M} = \{M_v\}_{v \in V(G')}$ where each M_v is a connected subset of $V(H')$. (Recall that G denotes the graph of the MULTICOLORED CLIQUE instance.)

1. For each $i \in [k]$ and $s \in [p]$, construct a vertex z_s^i with model

$$M_{z_s^i} := \bigcup_{j \in [k], j \neq i} \left(\left\{ x_0^{(i,j)}, \dots, x_{s-1}^{(i,j)} \right\} \cup \left\{ y_0^{(i,j)}, \dots, y_{p-s}^{(i,j)} \right\} \right). \quad (9.2)$$

2. For each edge $v_s^i v_t^j \in E(G)$ for $s, t \in [p]$ and $1 \leq i < j \leq k$, construct a vertex $r_{s,t}^{(i,j)}$ with:

$$M_{r_{s,t}^{(i,j)}} := \left\{ x_s^{(i,j)}, \dots, x_{p+1}^{(i,j)} \right\} \cup \left\{ y_{p-s+1}^{(i,j)}, \dots, y_{p+1}^{(i,j)} \right\} \quad (9.3)$$

$$\cup \left\{ x_t^{(j,i)}, \dots, x_{p+1}^{(j,i)} \right\} \cup \left\{ y_{p-t+1}^{(j,i)}, \dots, y_{p+1}^{(j,i)} \right\}. \quad (9.4)$$

Throughout the following, for $i \in [k]$ and $1 \leq i < j \leq k$, we use the notation

$$Z(i) := \bigcup_{s \in [p]} \{z_s^i\} \text{ and } R(i,j) := \bigcup_{\substack{s,t \in [p], \\ v_s^i v_t^j \in E(G)}} \{r_{s,t}^{(i,j)}\}, \quad (9.5)$$

respectively. We now observe the crucial property of G' .

Observation 9.3 (Claim 18 in [127]). For every $1 \leq i < j \leq k$, a vertex $z_h^i \in V(G')$ (a vertex $z_h^j \in V(G')$) is *not* adjacent to a vertex $r_{s,t}^{(i,j)} \in V(G')$ corresponding to the edge $v_s^i v_t^j \in E(G)$ if and only if $h = s$ ($h = t$).

Let $k' := k(k+1)/2$, then (G', k') is the instance resulting from this reduction. Suppose that G has a clique on vertices $\{v_{h_1}^1, \dots, v_{h_k}^k\}$, then using Observation 9.3, we can argue that the set

$$I := \{z_{h_1}^1, \dots, z_{h_k}^k\} \cup \{r_{h_i, h_j}^{(i,j)} \mid 1 \leq i < j \leq k\} \quad (9.6)$$

is an independent set of size k' in G' . Conversely, suppose that G' contains an independent set I of size k' . Since each $Z(i)$ and each $R(i,j)$ is a clique in G' , we can conclude that for each $i \in [k]$, there is a unique vertex $z_{h_i}^i \in Z(i) \cap I$, and for each $1 \leq i < j \leq k$, there is a unique vertex $r_{s_i, s_j}^{(i,j)} \in R(i,j) \cap I$. Since $r_{s_i, s_j}^{(i,j)}$ is neither adjacent to $z_{h_i}^i$ nor to $z_{h_j}^j$, we have by Observation 9.3 that $h_i = s_i$ and $h_j = s_j$. Therefore, $v_{h_i} v_{h_j} \in E(G)$, and since this holds for all i and j , $\{v_{h_1}^1, \dots, v_{h_k}^k\}$ is a clique in G . \square

9.2.2 Dominating Set

We now sketch the reduction due to Fomin et al. that proves the following result.

Theorem 9.4 (Fomin et al. [127]). DOMINATING SET is W[1]-hard on H-graphs parameterized by the number of edges in H plus solution size, and the hardness holds even if an H-representation of the input graph is given.

Proof (Sketch). Let G be an instance of MULTICOLORED INDEPENDENT SET with partition V_1, \dots, V_k of $V(G)$. Again we can assume that $k \geq 3$ and that $|V_i| = p$ for all $i \in [k]$. If the latter condition does not hold, let $p := \max_{i \in [k]} |V_i|$ and for each $i \in [k]$, add $p - |V_i|$ vertices to V_i that are adjacent to all vertices in each V_j where $j \neq i$. It is clear that the resulting instance has a multicolored independent set if and only if the original instance does.

The graph G' of the MINIMUM DOMINATING SET instances is obtained as follows. We take the graph G'' as constructed in the proof of Theorem 9.2, and for each $i \in [k]$, we add a vertex b_i whose model is $\{u_i\}$, i.e. it is adjacent to all vertices in $Z(i)$ and nothing else. We argue that G has a multicolored independent set if and only if G' has a dominating set of size k .

For the forward direction, if $I := \{v_{h_1}^1, \dots, v_{h_k}^k\}$ is a multicolored independent set in G , then using Observation 9.3, one can verify that $D := \{z_{h_1}^1, \dots, z_{h_k}^k\}$ is a dominating set in G' : Clearly, for each $i \in [k]$, the vertices in $Z(i) \cup \{b_i\}$ are dominated by $z_{h_i}^i \in D$. Suppose there is a vertex $r_{s,t}^{(i,j)} \in R(i,j)$ that is not dominated by D , then in particular it is neither adjacent to $z_{h_i}^i$ nor to $z_{h_j}^j$. By Observation 9.3, this implies that G contains the edge $v_{h_i}^i v_{h_j}^j$, a contradiction with the fact that I is an independent set.

For the backward direction, suppose that G' has a dominating set D of size k . Due to the vertices b_i (for $i \in [k]$), we can conclude that for all $i \in [k]$, $D \cap (Z(i) \cup \{b_i\}) \neq \emptyset$. If D contains b_i for some $i \in [k]$, then we can replace b_i by any vertex in $Z(i)$ such that the resulting set is still a dominating set of D , so we can assume that $D = \{z_{h_1}^1, \dots, z_{h_k}^k\}$. We claim that $\{v_{h_1}^1, \dots, v_{h_k}^k\}$ is an independent set in G . Suppose that for $i, j \in [k]$, there is an edge $v_{h_i}^i v_{h_j}^j \in E(G)$. Observation 9.3 implies that $r_{h_i, h_j}^{(i,j)}$ is neither adjacent to $z_{h_i}^i$ nor to $z_{h_j}^j$, so $r_{h_i, h_j}^{(i,j)}$ is not dominated by D , a contradiction. \square

The above reduction in fact does not only show hardness of the DOMINATING SET problem, but for all MIN- (σ^*, ρ^*) DOMINATION problems, when $0 \in \sigma^*$ and $\{1, 2\} \subseteq \rho^*$.

Corollary 9.5 (Fomin et al. [127]). For $\sigma^* \subseteq \mathbb{N}$ with $0 \in \sigma^*$ and $\rho^* \subseteq \mathbb{N}^+$ with $\{1, 2\} \subseteq \rho^*$, MIN- (σ^*, ρ^*) DOMINATION is W[1]-hard on H-graphs parameterized by the number of edges in H plus solution size, and the hardness holds even when an H-representation of the input graph is given.

σ	ρ	Standard name	W[1]-Hard
{0}	\mathbb{N}	Independent set	\circ_{\max}
\mathbb{N}	\mathbb{N}^+	Dominating set	\circ_{\min}
{0}	\mathbb{N}^+	Maximal Independent set	\circ_{\min}
\mathbb{N}^+	\mathbb{N}^+	Total Dominating set	\star_{\min}
{0}	{0, 1}	Strong Stable set or 2-Packing	
{0}	{1}	Perfect Code or Efficient Dom. set	
{0, 1}	{0, 1}	Total Nearly Perfect set	
{0, 1}	{1}	Weakly Perfect Dominating set	
{1}	{1}	Total Perfect Dominating set	
{1}	\mathbb{N}	Induced Matching	\star_{\max}
{1}	\mathbb{N}^+	Dominating Induced Matching	$\star_{\max}, \star_{\min}$
\mathbb{N}	{1}	Perfect Dominating set	
\mathbb{N}	{d, d + 1, ...}	d-Dominating set	\star_{\min}
{d}	\mathbb{N}	Induced d-Regular Subgraph	\star_{\max}
{d, d + 1, ...}	\mathbb{N}	Subgraph of Min Degree $\geq d$	
{0, 1, ..., d}	\mathbb{N}	Induced Subg. of Max Degree $\leq d$	\star_{\max}

Table 9.1: Some vertex subset properties expressible as (σ, ρ) -sets, with $\mathbb{N} = \{0, 1, \dots\}$ and $\mathbb{N}^+ = \{1, 2, \dots\}$. For each problem, at least one of the minimization, the maximization and the existence problem is NP-complete. For problems marked with \star_{\max} (\star_{\min}) in the right-most column, W[1]-hardness of the maximization (minimization) problem parameterized by mim-width + solution size is shown in the present paper. For problems marked with \circ_{\max} (\circ_{\min}) the W[1]-hardness of maximization (minimization) in the same parameterization was shown by Fomin et al. [127].

9.3 W[1]-Hard (σ, ρ) -Domination Problems by Mim-Width

Recall that INDEPENDENT SET and DOMINATING SET can be stated in the (σ, ρ) -framework as MAX-($\{0\}, \mathbb{N}$) and MIN-(\mathbb{N}, \mathbb{N}^+), respectively. In this section we adapt the reductions from the previous section to prove W[1]-hardness for several other problems expressible in the (σ, ρ) -framework, for an overview see Table 9.1.

9.3.1 Maximization Problems

The reductions given in this section are based on the reduction from MULTICOLORED CLIQUE to INDEPENDENT SET on H -graphs due to Fomin et al. [127], which we described in Section 9.2.1.

The following result states that a large class of (σ, ρ) maximization problems that are related to the INDEPENDENT SET problem according to their (σ, ρ) formulation are W[1]-hard on H -graphs parameterized by $\|H\|$ plus solution size. These problems include INDUCED MATCHING, DOMINATING INDUCED MATCHING, INDUCED

d -REGULAR SUBGRAPH, and INDUCED SUBGRAPH OF MAXIMUM DEGREE d , see Table 9.1.

Theorem 9.6. *For any fixed $d \in \mathbb{N}$ and $x \leq d + 1$, the following holds. Let $\sigma^* \subseteq \mathbb{N}_{\leq d}$ with $d \in \sigma^*$. Then, MAX- $(\sigma^*, \mathbb{N}_{\geq x})$ DOMINATION is W[1]-hard on H -graphs parameterized by the number of edges in H plus solution size, and the hardness holds even if an H -representation of the input graph is given.*

Proof. To prove the theorem, we provide a reduction from MULTICOLORED CLIQUE where given a graph G and a partition V_1, \dots, V_k of $V(G)$, the question is whether G contains a clique of size k using precisely one vertex from each V_i ($i \in [k]$). This problem is known to be W[1]-complete [119, 244].

Let (G, V_1, \dots, V_k) be an instance of MULTICOLORED CLIQUE. We can assume that $k \geq 3$ and that $|V_i| = p$ for $i \in [k]$. If the second assumption does not hold, let $p := \max_{i \in [k]} |V_i|$ and add $p - |V_i|$ isolated vertices to V_i , for each $i \in [k]$. (Note that adding isolated vertices does not change the answer to the problem.) For $i \in [k]$, we denote by v_1^i, \dots, v_p^i the vertices of V_i .

Let H be the graph constructed in Section 9.2.1, and let G'' denote the H -graph constructed in that section. We now describe how to obtain from G'' a graph G' that will be the graph of the instance of MAX- $(\sigma^*, \mathbb{N}_{\geq x})$ DOMINATION, by adding a gadget attached to each set $Z(i)$ and $R(i, j)$ (for all $1 \leq i < j \leq k$).

The New Gadget and the Construction of G' . Let X be a set of vertices of a graph. The gadget $\mathfrak{B}(X)$ is a complete bipartite graph on $2d - 1$ vertices and bipartition $(\{\beta_{1,1}, \dots, \beta_{1,d}\}, \{\beta_{2,1}, \dots, \beta_{2,d-1}\})$ such that for $h \in [d]$, each vertex $\beta_{1,h}$ is additionally adjacent to each vertex in X .

The graph G' is obtained from G'' by adding the gadgets $\mathfrak{B}(Z(i))$ for all $i \in [k]$ and the gadgets $\mathfrak{B}(R(i, j))$ for all $1 \leq i < j \leq k$. To prove the theorem, we require G' to be a K -graph for some graph K whose number of edges is bounded by a function of k , and possibly d , as d is fixed. We will show that G' is a K -graph for some supergraph K of H that meets this requirement.

Motivated by Observation 4.7 (see page 70), stating that each graph H is an H -graph, and the fact that the bipartite graph in each gadget $\mathfrak{B}(\cdot)$ has $\mathcal{O}(d^2)$ edges, we do the following to obtain K from H : For each $i \in [k]$, we add the gadget $\mathfrak{B}(\{u_i\})$, which will be used to encode $\mathfrak{B}(Z(i))$ in G' ; furthermore, for each $1 \leq i < j \leq k$, we add the gadget $\mathfrak{B}(\{w_{(i,j)}\})$, which will be used to encode $\mathfrak{B}(R(i, j))$ in G' . For an illustration of K , see Figure 9.2. We obtain a subdivision K' of K as follows:

- (K1) For all edges in $E(K) \cap E(H)$, we do the same subdivisions that were made to obtain H' from H .
- (K2) For each gadget $\mathfrak{B}(\cdot)$, we perform the edge subdivisions due to Observation 4.7 (see page 70) that allow for encoding the bipartite graph in $\mathfrak{B}(\cdot)$ as a $\mathfrak{B}(\cdot)$ -graph.

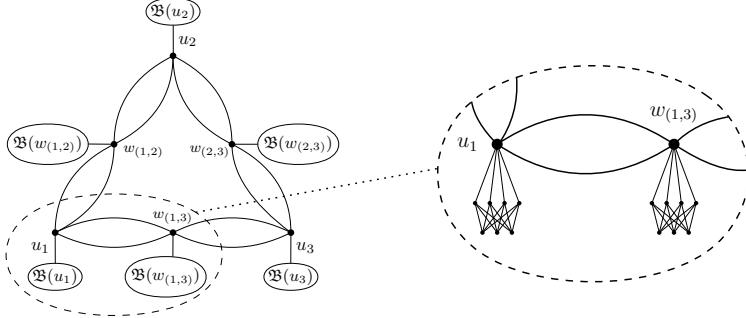


Figure 9.2: The graph K with respect to which the graph G' constructed in the proof of Theorem 9.6 is a K -graph. In this example, we have $k = 3$ and $d = 4$.

- (K3) For each $i \in [k]$, let $\{\beta_{1,1}^i, \dots, \beta_{1,d}^i, \beta_{2,1}^i, \dots, \beta_{2,d-1}^i\}$ be the vertices of $\mathfrak{B}(\{u_i\})$. Then, for each $h \in [d]$, we subdivide the edge $u_i \beta_{1,h}^i$, and denote the corresponding subdivision node by $s(i, h)$.
- (K4) Similarly, for each $1 \leq i < j \leq k$, let $\{\beta_{1,1}^{(i,j)}, \dots, \beta_{1,d}^{(i,j)}, \beta_{2,1}^{(i,j)}, \dots, \beta_{2,d-1}^{(i,j)}\}$ be the vertices of $\mathfrak{B}(\{w_{(i,j)}\})$. Then, for each $h \in [d]$, we subdivide the edge $u_i \beta_{1,h}^{(i,j)}$, and denote the corresponding subdivision node by $s((i, j), h)$.

We now give the K -representation of G' , $\mathcal{M}' = \{M'_v\}_{v \in V(G')}$, where each M'_v is a connected subset of $V(K')$; for an illustration of G' see Figure 9.3.

- (R1) For each vertex $v \in V(G') \cap V(G'')$, we let $M'_v := M_v$, where M_v is the model of v defined given in the construction of Fomin et al. which we described above. Note that each such vertex is either some vertex z_s^i or some vertex $r_{s,t}^{(i,j)}$ for appropriate choices for i, j, s , and t .
- (R2) For each $i \in [k]$, let $\{\beta_{1,1}^i, \dots, \beta_{1,d}^i, \beta_{2,1}^i, \dots, \beta_{2,d-1}^i\}$ be the vertices of $\mathfrak{B}(\{u_i\})$. By the subdivisions we did in Step K2, we obtain a corresponding complete bipartite graph on vertices $\{b_{1,1}^i, \dots, b_{1,d}^i, b_{2,1}^i, \dots, b_{2,d-1}^i\}$. Each $b_{t,h}^i$, where $t \in [2]$ and $h \in [d]$ if $t = 1$ and $h \in [d - 1]$ if $t = 2$, comes with a model from the subdivision of the complete bipartite graph of $\mathfrak{B}(\{u_i\})$, which we initially use as $M'_{b_{t,h}^i}$.
- (R3) We proceed analogously to Step R2 with each $\mathfrak{B}(\{w_{(i,j)}\})$, where $1 \leq i < j \leq k$.
- (R4) For each $i \in [k]$, and each $h \in [d]$, we add the node $s(i, h)$ to the models of z_s^i for all $s \in [p]$, and we add $s(i, h)$ to the model (in \mathcal{M}') of $b_{1,h}^i$. (This ensures that each $b_{1,h}^i$ is complete to $Z(i)$.)

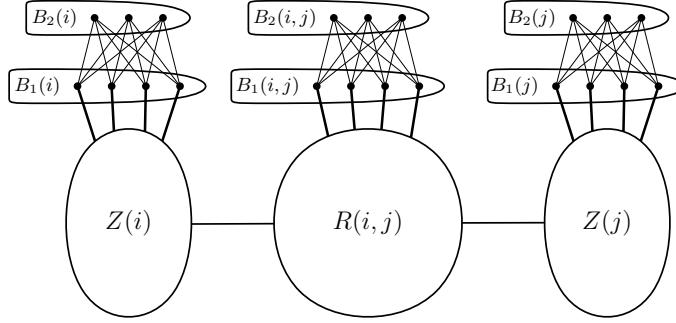


Figure 9.3: A part of the graph G' , where $1 \leq i < j \leq k$ and $d = 4$.

- (R5) For each $1 \leq i < j \leq k$ and $s, t \in [p]$ such that $v_s^i v_t^j \in E(G)$, and each $h \in [d]$, we add the node $s((i, j), h)$ to the model of $r_{s,t}^{(i,j)}$, and we add $s((i, j), h)$ to the model (in \mathcal{M}') of $b_{1,h}^{(i,j)}$. (This ensures that each $b_{1,h}^{(i,j)}$ is complete to $R(i, j)$.)

We count the size of K . For $|K|$, we observe that $|H| = k + \binom{k}{2}$ and each gadget $\mathfrak{B}(\cdot)$ has $2d - 1$ nodes, and we add $k + \binom{k}{2}$ such gadgets. Hence, $|K| = 2d \left(k + \binom{k}{2} \right) = dk(k + 1)$. As for $\|K\|$, we observe that the number of edges in H is $4 \cdot \binom{k}{2}$ and each gadget $\mathfrak{B}(\cdot)$ introduces $d(d - 1) + d = d^2$ edges. Hence,

$$\|K\| = 4 \cdot \binom{k}{2} + d^2 \left(k + \binom{k}{2} \right) = \mathcal{O}(d^2 \cdot k^2). \quad (9.7)$$

We introduce some more notation. For $1 \leq i < j \leq k$, we let

- $B_1(i) := \{b_{1,1}^i, \dots, b_{1,d}^i\}$, $B_2(i) := \{b_{2,1}^i, \dots, b_{2,d-1}^i\}$,
- $B_1(i, j) := \{b_{1,1}^{(i,j)}, \dots, b_{1,d}^{(i,j)}\}$ and $B_2(i, j) := \{b_{2,1}^{(i,j)}, \dots, b_{2,d-1}^{(i,j)}\}$; furthermore
- $B(i) := B_1(i) \cup B_2(i)$, $B(i, j) := B_1(i, j) \cup B_2(i, j)$, and
- $B := \bigcup_{i \in [k]} B(i) \cup \bigcup_{1 \leq i < j \leq k} B(i, j)$.

Note that $|B| = (2d - 1)(k + \binom{k}{2})$. We furthermore use the notation

$$Z_{+B}(i) := Z(i) \cup B(i) \text{ and } R_{+B}(i, j) := R(i, j) \cup B(i, j).$$

We now turn to the correctness proof of the reduction. We let $k' := 2d \cdot (k + \binom{k}{2})$ and show that G has a multicolored clique if and only if G' has a $(\sigma^*, \mathbb{N}_{\geq x})$ set of size k' . We first prove the forward direction. Note that the following claim yields the forward direction of the correctness proof, since a $(\{d\}, \{d + 1, \dots, d + k\})$ set is a $(\sigma^*, \mathbb{N}_{\geq x})$ set. (Recall that $d \in \rho^*$ and $x \leq d + 1$.)

Claim 9.6.1. If G has a multicolored clique, then G' has a $(\{d\}, \{d+1, \dots, d+k\})$ set of size $k' = 2d \cdot (k + \binom{k}{2})$ (assuming $k \geq 3$).

Proof. Let $\{v_{h_1}^1, \dots, v_{h_k}^k\}$ be the vertex set in G that induces the multicolored clique. By Observation 9.3 we can verify that

$$I := \{z_{h_1}^1, \dots, z_{h_k}^k\} \cup \left\{ r_{h_i, h_j}^{(i,j)} \mid 1 \leq i < j \leq k \right\} \quad (9.8)$$

is an independent set in G' . We let $S := I \cup B$ and observe that S is a $(\{d\}, \{d+1, \dots, d+k\})$ set: By construction, there is no edge between any pair of distinct sets of $B(i), B(i'), B(i, j), B(i', j')$, for any choice of $1 \leq i < j \leq k$ and $1 \leq i' < j' \leq k$.

Consider any vertex $x \in S$ and suppose that $x \in Z_{+B}(i)$ for some $i \in [k]$. (The case when $x \in R_{+B}(i, j)$ can be argued for analogously.) If $x = z_{h_i}^i$, then x is adjacent to the d vertices $b_{1,1}^i, \dots, b_{1,d}^i$, if $x = b_{1,\ell}^i$ for some $\ell \in [d]$, then x is adjacent to $z_{h_i}^i$ and the vertices $b_{2,1}^i, \dots, b_{2,d-1}^i$ and if $x = b_{2,\ell'}^i$ for some $\ell' \in [d-1]$, then it is adjacent to the vertices $b_{1,1}^i, \dots, b_{1,d}^i$. Hence, in all cases, x has precisely d neighbors in S .

Let $y \in V(G') \setminus S$ and note that $(V(G') \setminus S) \cap B = \emptyset$. If $y \in Z(i)$ for some $i \in [k]$, then $N(y) \cap S \supseteq \{z_{h_i}^i, b_{1,1}^i, \dots, b_{1,d}^i\}$, so $|N(y) \cap S| \geq d+1$. Since the only additional neighbors of y in S are in the set $R_i := \bigcup_{1 \leq j < i} R(j, i) \cup \bigcup_{i < j \leq k} R(i, j)$ and $R_i \cap S \subseteq I$, we can conclude that $|N(y) \cap (S \setminus B)| \leq k-1$, since I contains precisely one vertex from each set $R(i, j)$. We have argued that $d+1 \leq |N(y) \cap S| \leq d+k$. If $y \in R(i, j)$ for some $1 \leq i < j \leq k$, we can argue as before that $|N(y) \cap S| \geq d+1$ and since all neighbors of y in $S \setminus B(i, j)$ are contained either in $Z(i)$ or $Z(j)$, we can conclude that $d+1 \leq |N(y) \cap S| \leq d+3 \leq d+k$.

It remains to count the size of S . Clearly, $|I| = k + \binom{k}{2}$ and as observed above, $|B| = (2d-1)(k + \binom{k}{2})$, so

$$|S| = |I| + |B| = k + \binom{k}{2} + (2d-1) \left(k + \binom{k}{2} \right) = 2d \left(k + \binom{k}{2} \right) = k',$$

as claimed. \square

We now prove the backward direction of the correctness of the reduction. We begin by making several observations about the structure of $(\sigma^*, \mathbb{N}_{\geq x})$ sets in the graph G' .

Claim 9.6.2. Let $1 \leq i < j \leq k$.

- (i) Any $(\sigma^*, \mathbb{N}_{\geq x})$ set in G' contains at most $d+1$ vertices from each $Z(i) \cup B_1(i)$ or $R(i, j) \cup B_1(i, j)$.
- (ii) Any $(\sigma^*, \mathbb{N}_{\geq x})$ set contains at most $2d$ vertices from each $Z_{+B}(i)$ or $R_{+B}(i, j)$.
- (iii) If a $(\sigma^*, \mathbb{N}_{\geq x})$ set S contains $2d$ vertices from some $Z_{+B}(i)$ ($R_{+B}(i, j)$), then it contains at least one vertex from $Z(i)$ ($R(i, j)$) and each such vertex in $S \cap Z(i)$ ($S \cap R(i, j)$) has at least d neighbors in $S \cap Z_{+B}(i)$ ($S \cap R_{+B}(i, j)$).

Proof. (i) We prove the claim for a set $Z(i) \cup B_1(i)$. The proof for a set $R(i, j) \cup B_1(i, j)$ works analogously. Suppose for the sake of a contradiction that there is a set $S \subseteq V(G')$ that contains at least $d + 2$ vertices from some $Z(i) \cup B_1(i)$. Since $|B_1(i)| = d$, we know that S contains a vertex from $Z(i)$, say x . However, by construction, all vertices in $S \cap (Z(i) \cup B_1(i)) \setminus \{x\}$ are adjacent to x , implying that x has at least $d + 1$ neighbors in S , a contradiction with the fact that S is a $(\sigma^*, \mathbb{N}_{\geq x})$ set.

(ii) follows as a direct consequence, since $Z_{+B}(i) \setminus (Z(i) \cup B_1(i)) = B_2(i)$ and $|B_2(i)| = d - 1$. Similar for $R_{+B}(i, j)$.

(iii). The claim that S contains at least one vertex from $Z(i)$ is immediate since $|S \cap Z_{+B}(i)| = 2d$ and $|Z_{+B}(i) \setminus Z(i)| = |B(i)| = 2d - 1$. Let $x \in S \cap Z(i)$ be such a vertex. Then, the only vertices of $Z_{+B}(i)$ that x is *not* adjacent to are the vertices $B_2(i)$. Since $|B_2(i)| = d - 1$, the remaining vertices in $(S \cap Z_{+B}(i)) \setminus (B_2(i) \cup \{x\})$, of which there are at least d as we just argued, are neighbors of x . Similar for $R_{+B}(i, j)$. \square

Equipped with the previous claim, we can now finish the correctness proof of the reduction.

Claim 9.6.3. *If G' contains a $(\sigma^*, \mathbb{N}_{\geq x})$ set S of size $k' = 2d(k + \binom{k}{2})$, then G contains a multicolored clique.*

Proof. Let S be a $(\sigma^*, \mathbb{N}_{\geq x})$ set of size k' in G' . By Claim 9.6.2(ii), we can conclude that S contains precisely $2d$ vertices from each $Z_{+B}(i)$ and each $R_{+B}(i, j)$ (where $1 \leq i < j \leq k$). Consider any pair i, j with $1 \leq i < j \leq k$. By Claim 9.6.2(iii) we know that there are vertices

$$z_{s_i}^i \in Z(i) \cap S, \quad z_{s_j}^j \in Z(j) \cap S, \quad \text{and } r_{t_i, t_j}^{(i, j)} \in R(i, j) \cap S,$$

for some $s_i, s_j, t_i, t_j \in [p]$. Again by Claim 9.6.2(iii), $z_{s_i}^i$ has d neighbors in $Z_{+B}(i) \cap S$, so if $z_{s_i}^i r_{t_i, t_j}^{(i, j)} \in E(G')$, then $z_{s_i}^i$ has $d + 1$ neighbors in S , a contradiction with the fact that S is a $(\sigma^*, \mathbb{N}_{\geq x})$ set. Hence, $z_{s_i}^i r_{t_i, t_j}^{(i, j)} \notin E(G')$ and $z_{s_j}^j r_{t_i, t_j}^{(i, j)} \notin E(G')$. By Observation 9.3, we then have that $s_i = t_i$ and $s_j = t_j$. We can conclude that $v_{s_i}^i v_{s_j}^j \in E(G)$ and since the argument holds for any pair of indices i, j , G has a multicolored clique. \square

We would like to remark that by the proof of the previous claim, we have established that any $(\sigma^*, \mathbb{N}_{\geq x})$ set S in G' of size k' in fact contains all vertices from B and one vertex from each $Z(i)$ and from each $R(i, j)$. Since this is precisely the shape of the set constructed in the forward direction of the correctness proof, this shows that any $(\sigma^*, \mathbb{N}_{\geq x})$ set of size k' in G' is a $(\{d\}, \{d + 1, \dots, d + k\})$ set (assuming $k \geq 3$).

Claims 9.6.1 and 9.6.3 establish the correctness of the reduction. We observe that $|V(G')| = \mathcal{O}(|V(G)| + d^2 \cdot k^2)$ and clearly, G' can be constructed from G in time polynomial in $|V(G)|$, d and k as well. Furthermore, by (9.7), $\|K\| = \mathcal{O}(d^2 \cdot k^2)$ which implies that $\|K\| = \mathcal{O}(k^2)$ since d is a fixed constant and the theorem follows. \square

By Theorem 4.27, the previous theorem has the following consequence.

Corollary 9.7. *For any fixed $d \in \mathbb{N}$ and $x \leq d+1$, the following holds. Let $\sigma^* \subseteq \mathbb{N}_{\leq d}$ with $d \in \sigma^*$. Then, MAX- $(\sigma^*, \mathbb{N}_{\geq x})$ DOMINATION is W[1]-hard parameterized by linear mim-width plus solution size, and the hardness holds even if a corresponding decomposition tree is given.*

9.3.2 Minimization Problems

In this section we prove W[1]-hardness of minimization versions of several (σ, ρ) problems parameterized by linear mim-width plus solution size. These problems include TOTAL DOMINATING SET, DOMINATING INDUCED MATCHING, and d -DOMINATING SET, see Table 9.1 on page 185 for details.

We obtain our results by modifying the reduction from MULTICOLORED INDEPENDENT SET to MINIMUM DOMINATING SET on H -graphs parameterized by solution size plus $\|H\|$ due to Fomin et al. [127] which we presented in Section 9.2.2. Recall that in the MULTICOLORED INDEPENDENT SET problem we are given a graph G and a partition V_1, \dots, V_k of its vertex set $V(G)$ and the question is whether there is an independent set $\{v_1, \dots, v_k\} \subseteq V(G)$ in G such that for each $i \in [k]$, $v_i \in V_i$. The W[1]-hardness of this problem follows immediately from the W[1]-hardness of the MULTICOLORED CLIQUE problem via taking the complement graph.

Total Domination Problems

Recall that the (σ, ρ) formulation for DOMINATING SET is $(\mathbb{N}, \mathbb{N}^+)$. We now explain how to modify the above reduction to obtain hardness for total dominating set problems where each vertex in the solution has to have at least one neighbor in the solution as well. These problems include TOTAL DOMINATING SET and DOMINATING INDUCED MATCHING, which can be formulated as $(\mathbb{N}^+, \mathbb{N}^+)$ and $(\{1\}, \mathbb{N}^+)$, respectively. The minimization problem of either of them is known to be NP-complete.

Theorem 9.8. *For $\sigma^* \subseteq \mathbb{N}^+$ with $1 \in \sigma^*$ and $\rho^* \subseteq \mathbb{N}^+$ with $\{1, 2\} \subseteq \rho^*$, MIN- (σ^*, ρ^*) DOMINATION is W[1]-hard on H -graphs parameterized by the number of edges in H plus solution size, and the hardness holds even when an H -representation of the input graph is given.*

Proof. We modify the above reduction from MULTICOLORED INDEPENDENT SET as follows. For each $i \in [k]$, we add another vertex c_i to G' which is only adjacent to b_i . We let $B := \bigcup_{i \in [k]} \{b_i\}$ and $C := \bigcup_{i \in [k]} \{c_i\}$. Note that these new vertices can be ‘hardcoded’ into H with the number of edges in H increasing only by k . To argue the correctness of the reduction, we now show that G has a multicolored independent set if and only if G' has a (σ^*, ρ^*) set of size $k' := 2k$.

For the forward direction, suppose that G has an independent set $\{v_{h_1}^1, \dots, v_{h_k}^k\}$. Then, $D' := \{z_{h_1}^1, \dots, z_{h_k}^k\}$ dominates all vertices in $V(G') \setminus C$ by the same argument

as above and $D := D' \cup B$ dominates all vertices of G' . Furthermore, each $x \in D$ has precisely one neighbor in D : For each such x , either $x = z_{h_i}^i$ or $x = b_i$ for some $i \in [k]$. In the former case, $N(x) \cap D = \{b_i\}$ and in the latter case, $N(x) \cap D = \{z_{h_i}^i\}$. Now let $y \in V(G') \setminus D$. If $y \in Z(i) \cup \{c_i\}$ for $i \in [k]$, then $\emptyset \neq N(y) \cap D \subseteq \{z_{h_i}^i, b_i\}$. If $y \in R(i, j)$ for some $1 \leq i < j \leq k$, then y is either dominated by one of $z_{h_i}^i$ and $z_{h_j}^j$ or by both and it cannot have any other neighbors in D by construction. Since $1 \in \sigma^*$ and $\{1, 2\} \subseteq \rho^*$, D is a (σ^*, ρ^*) set and clearly, $|D| = 2k$.

For the backward direction, suppose that G' has a (σ^*, ρ^*) set D of size $2k$. Let $i \in [k]$. Since $0 \notin \sigma^*$ and $0 \notin \rho^*$, we have that at least one of c_i and b_i is contained in D (either c_i is dominating or it needs to be dominated). Suppose $c_i \in D$. Since each vertex in D has to have at least one neighbor in D and b_i is the only neighbor of c_i , we can conclude that $b_i \in D$. So, in either case, we have that b_i is contained in D and subsequently we have that $B \subseteq D$. Since $0 \notin \sigma^*$, all vertices of B have a neighbor in D . Suppose for some $i \in [k]$ that neighbor is c_i . Then, we can replace c_i with some $z_{h_i}^i \in Z(i)$, without changing the fact that D is a (σ^*, ρ^*) set. We can assume that for each $i \in [k]$, the neighbor of b_i that is contained in D is a vertex $z_{h_i}^i \in Z(i)$. We have that $D = B \cup \{z_{h_1}^1, \dots, z_{h_k}^k\}$ and since D is a dominating set (in other words, $0 \notin \rho^*$), we can again argue using Observation 9.3 that $\{v_{h_1}^1, \dots, v_{h_k}^k\}$ is an independent set in G . \square

d -Domination Problems

As a somewhat orthogonal result to Theorem 9.6, we now show hardness of several problems related to the d -DOMINATING SET problem, where each vertex that is not in the solution set has to be dominated by at least some fixed number of d neighbors in the solution. We use a similar gadget as the one constructed in the proof of Theorem 9.6 to prove hardness of several (σ, ρ) problems where each vertex has to be dominated by at least d vertices. In particular, we prove the following theorem. Note that the analogous statement of the following theorem for $d = 1$ is proved by the reduction explained in the beginning of this section, see Corollary 9.5.

Theorem 9.9. *For any fixed $d \in \mathbb{N}_{\geq 2}$, the following holds. Let $\sigma^* \subseteq \mathbb{N}$ with $\{0, 1, d - 1\} \subseteq \sigma^*$ and $\rho^* \subseteq \mathbb{N}_{\geq d}$ with $\{d, d + 1\} \subseteq \rho^*$. Then, MIN- (σ^*, ρ^*) DOMINATION is W[1]-hard on H -graphs parameterized by the number of edges in H plus solution size, and the hardness even holds when an H -representation of the input graph is given.*

Proof. We modify the reduction from MULTICOLORED INDEPENDENT SET to DOMINATING SET on H -graphs due to Fomin et al. [127] that we sketched in Section 9.2.2. Let G be a graph with vertex partition V_1, \dots, V_k and $|V_i| = p$ for all $i \in [k]$ and assume $k \geq 3$. We first describe the gadget we use and then we describe how to construct the graph G' of the MIN- (σ^*, ρ^*) DOMINATION instance.

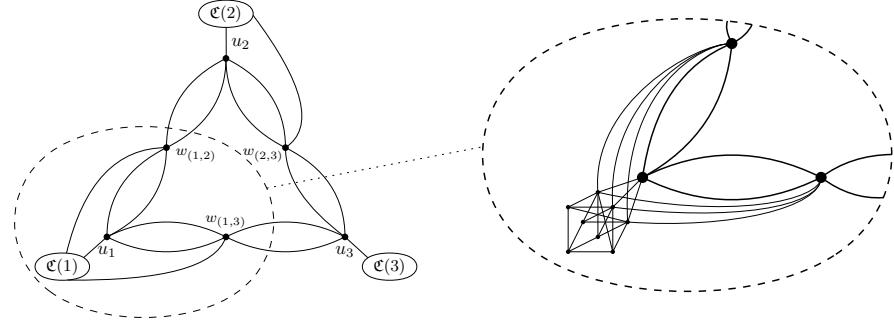


Figure 9.4: An example graph K w.r.t. which the graph G' constructed in the proof of Theorem 9.9 is a K -graph. In this example, $k = 3$.

The Gadget $\mathfrak{C}(i)$. Let $i \in [k]$. The gadget $\mathfrak{C}(i)$ is a complete bipartite graph with bipartition $(C_1(i), C_2(i))$ where $C_1(i) := \{c_{1,1}^i, \dots, c_{1,d}^i\}$ and $C_2(i) := \{c_{2,1}^i, \dots, c_{2,d}^i\}$ such that each vertex $c_{1,j}^i$ for $j \in [d-1]$ is additionally adjacent to all vertices in $Z(i)$ as well as to all vertices in $R(i, j)$ for $j > i$. (Note that $c_{1,d}^i$ does not have these additional adjacencies.) Throughout the following, we let $C(i) := C_1(i) \cup C_2(i)$ and $C := \bigcup_{i \in [k]} C(i)$.

The graph G' is now obtained by constructing the graph G'' as in the proof of Theorem 9.6 and then, for each $i \in [k]$, adding the gadget $\mathfrak{C}(i)$ and adding a ‘satellite vertex’ s_i , adjacent to all vertices in $Z(i) \cup C_1(i)$. G' is a K -graph for the graph $K \supseteq H$, obtained by ‘hardcoding’ each $\mathfrak{C}(i)$, for $i \in [k]$, into H . That is, for each $i \in [k]$, we add a complete bipartite graph with bipartition $(\{\gamma_{1,1}^i, \dots, \gamma_{1,d}^i\}, \{\gamma_{2,1}^i, \dots, \gamma_{2,d}^i\})$, and make all vertices $\gamma_{1,h}^i$, where $h \in [d-1]$, adjacent to u_i as well as to all vertices $w_{(i,j)}$ with $j > i$. For an illustration of K see Figure 9.4. Note that

$$\|K\| = \|H\| + k(d^2 + 1) + \sum_{i=1}^k (k-i)(d-1) = \mathcal{O}(k^2 \cdot d + k \cdot d^2). \quad (9.9)$$

We illustrate the structure of the graph G' in Figure 9.5. We now argue that G' is a K -graph. We begin by constructing a subdivision K' of K . First, we do Step K1 that was taken in the proof of Theorem 9.6 (see page 186) to construct the subdivision, and then the analogue of Step K2 in the proof of Theorem 9.6 for the gadgets $\mathfrak{C}(i)$. We continue with the following two steps.

- (K3) For each $i \in [k]$, let $\{\gamma_{1,1}^i, \dots, \gamma_{1,d}^i, \gamma_{2,1}^i, \dots, \gamma_{2,d-1}^i\}$ be the vertices of the copy of the graph of $\mathfrak{C}(i)$ in K . For each $h \in [d-1]$, we subdivide the edge $u_i \gamma_{1,h}^i$ once, and denote the corresponding subdivision node by $s(i, h)$.
- (K4) Furthermore, for each $i \in [k]$, for each $i < j \leq k$, and $h \in [d-1]$ we subdivide

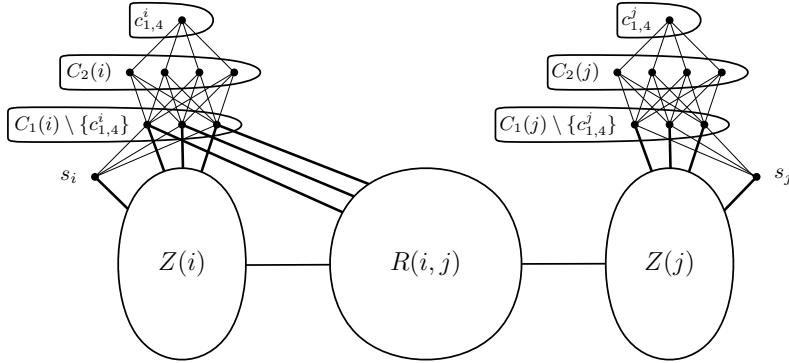


Figure 9.5: Illustration of a part of G' constructed in the proof of Theorem 9.9, where $1 \leq i < j \leq k$ and $d = 4$.

the edge $w_{(i,j)}\gamma_{1,h}^i$ once, and denote the corresponding subdivision node by $s((i,j),h)$.

We now sketch how to obtain a K -representation of G' , $\mathcal{M}' = \{M'_v\}_{v \in V(G')}$, where each M'_v is a connected subset of $V(K')$; for an illustration of G' see Figure 9.5. As these steps are very similar to Steps R1 to R5 in the proof of Theorem 9.6 (see page 187), we focus on pointing out how to adapt them rather than fully restating all of them.

First, for each $i \in [k]$, the model for the vertex s_i consists of the vertex u_i , i.e. we add the model $M'_{s_i} := \{u_i\}$ to \mathcal{M}' . Then we do the steps analogous to Steps R1 and R2 taken in the proof of Theorem 9.6. Following that, we take the steps analogous to R4 and R5, where in Step R4 we consider vertex $c_{1,h}^i$ instead of vertex $b_{1,h}^i$, and in Step R5, we consider vertex $c_{1,h}^i$ instead of vertex $b_{1,h}^{(i,j)}$. Furthermore, in Step R4, we additionally add $s(i,h)$ to the model of s_i . This completes the construction of the K -representation for G' .

Claim 9.9.1. *If G has a multicolored independent set, then G' has a (σ^*, ρ^*) set of size $k' := k \cdot (d + 1)$.*

Proof. Let $\{v_{h_1}^1, \dots, v_{h_k}^k\}$ be the independent set in G . By the reduction proving Proposition 9.5 in the beginning of this section, we have that $D' := \{z_{h_1}^1, \dots, z_{h_k}^k\}$ is a $(\{0\}, \{1, 2\})$ -set of $G' - C$ of size k . Let $C_1 := \bigcup_{i \in [k]} C_1(i)$, $C_2 := C \setminus C_1$ and $D := D' \cup C_1$.

Since each vertex in $V(G') \setminus (D \cup C)$ is adjacent to precisely $d - 1$ vertices in C_1 and to either one or two vertices in D' (and $D' \cap C_1 = \emptyset$), we can conclude that each vertex in $V(G') \setminus (D \cup C)$ is adjacent to either d or $d + 1$ vertices in D . Since each $C(i)$ induces a $K_{d,d}$, we can conclude that all vertices in C_2 have d neighbors in D as

well. Furthermore, $N(s_i) \cap D = (C_1(i) \setminus \{c_{1,d}^i\}) \cup \{z_{h_i}^i\}$, so we have that all vertices in G' that are not contained in D have either d or $d+1$ neighbors in D .

Let $i \in [k]$. Then, $N(z_{h_i}^i) \cap D = \{c_{1,1}^i, \dots, c_{1,d-1}^i\}$, $N(c_{1,d}^i) \cap D = \emptyset$ and for $\ell \in [d-1]$, $N(c_{1,\ell}^i) \cap D = \{z_{h_i}^i\}$. We can conclude that D is a $(\{0, 1, d-1\}, \{d, d+1\})$ -set in G' and clearly, $|D| = k + kd = k'$. \square

In what follows, the strategy is to argue that each (σ^*, ρ^*) set of size $k' = k \cdot (d+1)$ contains a set $\{z_{h_1}^1, \dots, z_{h_k}^k\}$ which will imply that $\{v_{h_1}^1, \dots, v_{h_k}^k\}$ is an independent set in G . Throughout the following, for $i \in [k]$, we let $Z_+(i) := Z(i) \cup C(i) \cup \{s_i\}$.

Claim 9.9.2. *For all $i \in [k]$, any (σ^*, ρ^*) set D in G' contains at least d vertices from $C(i)$ and at least $d+1$ vertices from $Z_+(i)$.*

Proof. We first show that each such D contains at least d vertices from $C(i)$. Suppose not, then $|D \cap C(i)| \leq d-1$ for some $i \in [k]$. If $c_{1,d}^i \notin D$, then $C_2(i) \subseteq D$, otherwise $c_{1,d}^i$ cannot have d or more neighbors in D . But $|C_2(i)| = d$, a contradiction. We can assume that $c_{1,d}^i \in D$. Furthermore, there is at least one vertex $c_{2,\ell}^i$ for $\ell \in [d]$ with $c_{2,\ell}^i \notin D$. To ensure that $c_{2,\ell}^i$ has at least d neighbors in D , we would have to include all remaining vertices from $C_1(i)$ in D , but then $|D \cap C(i)| \geq d$, a contradiction. The second part of the claim now follows since the vertex s_i only has neighbors in $Z_+(i)$ and at most $d-1$ neighbors in $C(i) \cap D$ (namely $C_1(i) \setminus \{c_{1,d}^i\}$): Since D is a (σ^*, ρ^*) set, it either has to contain s_i or at least one additional neighbor of s_i . \square

Claim 9.9.3. *For all $i \in [k]$, any (σ^*, ρ^*) set D of size at most $k' = k(d+1)$ contains $C_1(i)$. We furthermore can assume that it additionally contains some $z_{h_i}^i \in Z(i)$, where $h_i \in [p]$.*

Proof. By Claim 9.9.2 we have that D contains $d+1$ vertices from each $Z_+(i')$, $i' \in [k]$, and no other vertices. Consider any vertex $z_s^i \in Z(i)$ (where $s \in [p]$) that is not contained in D . Recall that z_s^i has to have at least d neighbors in D . By Claim 9.9.2, z_s^i has precisely one neighbor in $(Z(i) \cup \{s_i\}) \cap D$ and since D does not contain any vertex from any $R(j, i)$ ($1 \leq j < i$) or $R(i, j')$ ($i < j' \leq k$), the only possible neighbors of z_s^i in D are $(C_1(i) \cup \{s_i\}) \setminus \{c_{1,d}^i\}$. Furthermore, we observe that $c_{1,d}^i \in D$: for if $c_{1,d}^i \notin D$, then $c_{1,d}^i$ has to have either d or $d+1$ neighbors in D . However, D already contains the $d-1$ vertices $C_1(i) \setminus \{c_{1,d}^i\}$ that are not adjacent to $c_{1,d}^i$, and D contains $d+1$ vertices from $Z_+(i)$. So, at most two neighbors of $c_{1,d}^i$ are contained in D . If at most one neighbor of $c_{1,d}^i$ is contained in D , this immediately gives a contradiction with D being a (σ^*, ρ^*) set since $d \geq 2$. However, if $d = 2$, and two neighbors of $c_{1,d}^i$ are contained in D , then each vertex in $Z(i)$ has only $d-1$ neighbors in D , namely the ones in $C_1(i) \setminus \{c_{1,d}^i\}$, again a contradiction with D being a (σ^*, ρ^*) set. We can conclude that $C_1(i) \subseteq D$.

Now suppose that $s_i \in D$. Then, after swapping s_i with any vertex in $Z(i)$, the resulting set remains a (σ^*, ρ^*) set: Clearly, the condition of being a (σ^*, ρ^*) set is

not violated by any vertex in $Z_+(i)$. For $i < j \leq k$, consider any vertex $x \in R(i, j)$. Then, $N(x) \cap D$ contains the $d - 1$ vertices $C_1(i) \setminus \{c_{1,d}^i\}$, and at most one more each from $Z(i)$ and $Z(j)$, as D can contain at most one vertex from each $Z(i')$, $i' \in [k]$. Now, if we swapped s_i with some vertex from $Z(i)$, then this means that initially, D contained d neighbors from $N(x)$, namely $C_1(i) \setminus \{c_{1,d}^i\}$ and one vertex from $Z(j)$. Hence, after swapping, D contains $d + 1$ vertices and since $d + 1 \in \rho^*$, D remained a (σ^*, ρ^*) set. An analogous argument can be given for any $R(j', i)$, where $1 \leq j' < i$. \square

We are now ready to conclude the correctness proof of the reduction.

Claim 9.9.4. *If G' has a (σ^*, ρ^*) set of size $k' = k(d + 1)$, then G has a multicolored independent set.*

Proof. Let D be a (σ^*, ρ^*) set of size k' . By Claim 9.9.3, we can assume that $D = C_1 \cup \{z_{h_1}^1, \dots, z_{h_k}^k\}$ for some $h_1, \dots, h_k \in [p]$. Now, since for each $1 \leq i < j \leq k$, all vertices in $R(i, j)$ have precisely $d - 1$ neighbors in C_1 , each of them has to have at least one of $z_{h_i}^i$ and $z_{h_j}^j$ as a neighbor. By Observation 9.3, this allows us to conclude that $\{v_{h_1}^1, \dots, v_{h_k}^k\}$ is an independent set in G . \square

Claims 9.9.1 and 9.9.4 establish the correctness of the reduction. Clearly, $|V(G')| = \mathcal{O}(|V(G)| + d^2 \cdot k)$ (and G' can be constructed in polynomial time) and by (9.9), $\|K\| = \mathcal{O}(k^2 \cdot d + k \cdot d^2)$. Since d is a fixed constant we have that $\|K\| = \mathcal{O}(k^2)$ and the theorem follows. \square

Similarly to above, a combination of the previous two theorems with Theorem 4.27 yields the following hardness results for (σ, ρ) minimization problems on graphs of bounded linear mim-width.

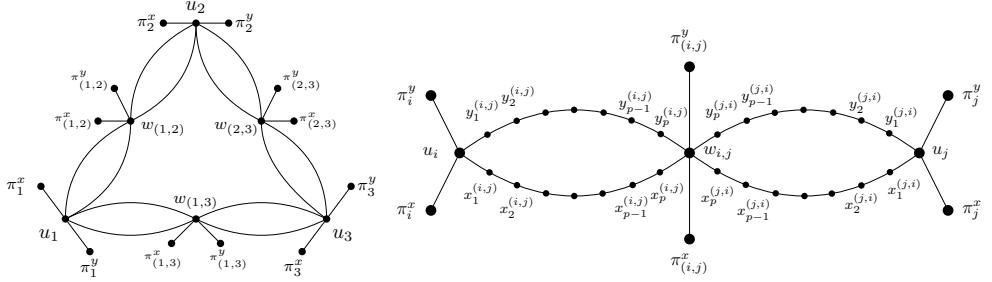
Corollary 9.10. *Let $\sigma^* \subseteq \mathbb{N}$ and $\rho^* \subseteq \mathbb{N}$. Then, MIN- (σ^*, ρ^*) DOMINATION is W[1]-hard parameterized by linear mim-width plus solution size, if one of the following holds.*

- (i) $\sigma^* \subseteq \mathbb{N}^+$ with $1 \in \sigma^*$ and $\rho^* \subseteq \mathbb{N}^+$ with $\{1, 2\} \subseteq \rho^*$.
- (ii) For some fixed $d \in \mathbb{N}_{\geq 2}$, $\{0, 1, d - 1\} \subseteq \sigma^*$ and $\rho^* \subseteq \mathbb{N}_{\geq d}$ with $\{d, d + 1\} \subseteq \rho^*$.

Furthermore, the hardness holds even if a corresponding decomposition tree is given.

9.4 FVS is W[1]-Hard by Mim-Width

We now prove that FEEDBACK VERTEX SET is W[1]-hard parameterized by mim-width, ruling out the possibility of FPT-algorithms for this parameterized problem under the standard assumption that FPT \neq W[1]. Again we will prove our results by considering the MAXIMUM INDUCED FOREST problem, the dual to FEEDBACK



(a) Illustration of the graph H for $k = 3$. (b) A part of the subdivision H' of H , where $1 \leq i < j \leq k$.

VERTEX SET. We prove that MAXIMUM INDUCED FOREST on H -graphs is W[1]-hard parameterized by solution size plus $\|H\|$, the number of edges in H , which together with Theorem 4.27 yields the hardness result replacing $\|H\|$ by linear mim-width in the parameterization.

Theorem 9.11. MAXIMUM INDUCED FOREST on H -graphs is W[1]-hard parameterized by $k + \|H\|$, where k denotes the solution size, and the hardness holds even when an H -representation of the input graph is given.

Proof. Let (G, V_1, \dots, V_k) be an instance of MULTICOLORED CLIQUE. We can assume that $k \geq 2$ and that $|V_i| = p$ for $i \in [k]$. If the second assumption does not hold, let $p := \max_{i \in [k]} |V_i|$ and add $p - |V_i|$ isolated vertices to V_i , for each $i \in [k]$; we denote by v_1^i, \dots, v_p^i the vertices of V_i .

We obtain an H -graph G' from an adapted version of the construction due to Fomin et al. presented in Section 9.2.1.

The graph H is obtained as follows, see Figure 9.6a for an illustration. Let K be the graph created in the reduction of Section 9.2.1, and assume the same names for all its vertices throughout the following; then H is obtained from K by taking the following two additional steps.

3. For each $i \in [k]$, add to H two neighbors π_i^x and π_i^y of u_i .
4. For each $1 \leq i < j \leq k$, add to H two neighbors $\pi_{(i,j)}^x$ and $\pi_{(i,j)}^y$ of $w_{(i,j)}$.

We let $\Pi := \bigcup_{i \in [k]} \{\pi_i^x, \pi_i^y\} \cup \bigcup_{1 \leq i < j \leq k} \{\pi_{(i,j)}^x, \pi_{(i,j)}^y\}$. Note that

$$\|H\| = (3/2)k(k+1) \text{ and } \|H\| = k(3k-1). \quad (9.10)$$

We obtain a subdivision H' of H by subdividing each edge in $E(G - \Pi) = E(K)$ in the same way as in Section 9.2.1. We illustrate this in Figure 9.6b.

We now construct the H -graph G' by defining its H -representation $\mathcal{M} = \{M_v\}_{v \in V(G')}$ where each M_v is a connected subset of $V(H')$, again this step is based on the construction given in Section 9.2.1. (Recall that G denotes the graph of the MULTICOLORED CLIQUE instance.)

1. For each $i \in [k]$ and $s \in [p]$, we add a vertex z_s^i (representing vertex v_s^i from G) whose model is obtained from the set shown in (9.2) and adding the vertices π_i^x and π_i^y .
2. For each $i \in [k]$, construct vertices α_i^x with model $M_{\alpha_i^x} := \{\pi_i^x\}$ and α_i^y with model $M_{\alpha_i^y} := \{\pi_i^y\}$.
3. For each edge $v_s^i v_t^j \in E(G)$ for $s, t \in [p]$ and $1 \leq i < j \leq k$, construct a vertex $r_{s,t}^{(i,j)}$ whose model is obtained from the set shown in (9.3) and (9.4), and adding the vertices $\pi_{(i,j)}^x$ and $\pi_{(i,j)}^y$.
4. For each $1 \leq i < j \leq k$, construct vertices $\alpha_x^{(i,j)}$ with model $M_{\alpha_x^{(i,j)}} := \{\pi_{(i,j)}^x\}$ and $\alpha_y^{(i,j)}$ with model $M_{\alpha_y^{(i,j)}} := \{\pi_{(i,j)}^y\}$.
5. Construct a vertex β with model $M_\beta := V(H) \setminus \Pi$.

Throughout the following, for $i \in [k]$ and $1 \leq i < j \leq k$, respectively, we use the notations $Z(i)$ and $R(i, j)$ introduced in (9.5), and we let $Z_{+\alpha}(i) := Z(i) \cup \{\alpha_x^i, \alpha_y^i\}$ and $R_{+\alpha}(i, j) := R(i, j) \cup \{\alpha_x^{(i,j)}, \alpha_y^{(i,j)}\}$. We furthermore define

$$A := \bigcup_{i \in [k]} \{\alpha_x^i, \alpha_y^i\} \cup \bigcup_{1 \leq i < j \leq k} \{\alpha_x^{(i,j)}, \alpha_y^{(i,j)}\}.$$

We continue with some observations about the global structure of G' .

Observation 9.11.1. Let $1 \leq i < j \leq k$ (wherever required).

- (i) $N(\alpha_x^i) = Z(i) = N(\alpha_y^i)$, $N(\alpha_x^{(i,j)}) = R(i, j) = N(\alpha_y^{(i,j)})$, $N(\beta) = V(G') \setminus A$.
- (ii) $Z(i)$ induces a clique in G' and $R(i, j)$ induces a clique in G' .
- (iii) A is an independent set in G' of size $2k + 2 \cdot \binom{k}{2}$.

By Observation 9.11.1, the structure of the graph G' can be illustrated as shown in Figure 9.7. The interactions between vertices in the sets $Z(\cdot)$ and $R(\cdot, \cdot)$ remains the same as in Section 9.2.1, therefore Observation 9.3 still holds in the context of the present reduction. We are now ready to prove the correctness of the reduction. In particular we will show that G has a multicolored clique if and only if G' has an induced forest of size $k' := 3k + 3 \binom{k}{2} + 1$.

Claim 9.11.2. If G has a multicolored clique on vertex set $\{v_{h_1}^1, \dots, v_{h_k}^k\}$, then G' has an induced forest of size $k' = 3k + 3 \cdot \binom{k}{2} + 1$.

Proof. Using Observation 9.3, one can easily verify that the set

$$I := \{z_{h_1}^1, \dots, z_{h_k}^k\} \cup \{r_{h_i, h_j}^{(i,j)} \mid 1 \leq i < j \leq k\} \tag{9.11}$$

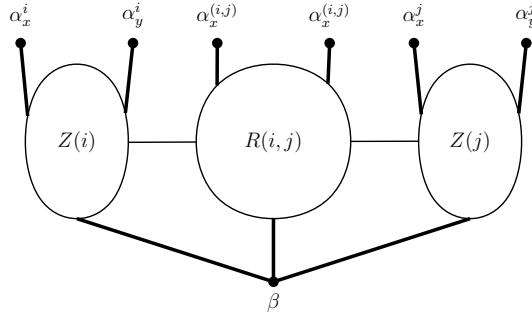


Figure 9.7: Illustration of a part of G' , where $1 \leq i < j \leq k$. Bold edges imply that all possible edges between the corresponding (sets of) vertices are present. Non-bold edges mean that *some* of the edges between the two sets of vertices are present, depending on the construction.

is an independent set in G' . By Observation 9.11.1(iii) and the construction given above, we can conclude that $F := I \cup A \cup \{\beta\}$ induces a forest in G' : I and A are both independent sets and $A \cup I$ induces a disjoint union of paths on three vertices, the middle vertices of which are contained in I . The only additional edges that are introduced are between β and vertices in I , so F induces a tree. Clearly, $|F| = |I| + |A| + |\{\beta\}| = k + \binom{k}{2} + 2k + 2 \cdot \binom{k}{2} + 1 = k'$, proving the claim. \square

We now prove the backward direction of the correctness of the reduction. This will be done by a series of claims and observations narrowing down the shape of any induced forest on k' vertices in G' . Eventually, we will be able conclude that any such induced forest contains an independent set of size $k + \binom{k}{2}$ of the shape (9.11). We can then conclude that G contains a multicolored clique by Observation 9.3.

The following is a direct consequence of Observation 9.11.1(ii).

Observation 9.11.3. Let F be an induced forest in G' . Then, $V(F)$ contains

- (i) at most 2 vertices from $Z(i)$, where $i \in [k]$ and
- (ii) at most 2 vertices from $R(i,j)$, where $1 \leq i < j \leq k$.

Next, we investigate the interaction of any induced forest with the sets $Z_{+\alpha}(i)$ and $R_{+\alpha}(i,j)$.

Claim 9.11.4. Let F be an induced forest in G' . If $V(F)$ contains two vertices from $Z(i)$, where $i \in [k]$ (from $R(i,j)$, where $1 \leq i < j \leq k$), then $V(F)$ cannot contain a vertex from $\{\alpha_x^i, \alpha_y^i\}$ (from $\{\alpha_x^{(i,j)}, \alpha_y^{(i,j)}\}$, respectively).

Proof. Suppose $V(F)$ contains two vertices $a, b \in Z(i)$. We prove the claim for α_x^i and note that the same holds for α_y^i . By Observation 9.11.1(ii), a and b are adjacent

and α_x^i is adjacent to both a and b by Observation 9.11.1(i). Hence, $\{\alpha_x^i, a, b\}$ induces a 3-cycle in G' .

An analogous argument can be given for the second statement. \square

In the light of Observation 9.11.3 and Claim 9.11.4, we make

Observation 9.11.5. *Let F be an induced forest in G' . If $V(F)$ contains three vertices from $Z_{+\alpha}(i)$ for some $i \in [k]$ (three vertices from $R_{+\alpha}(i, j)$, respectively), then this set of three vertices must include α_x^i and α_y^i (resp., $\alpha_x^{(i,j)}$ and $\alpha_y^{(i,j)}$).*

The previous observation implies that in G' , any induced forest on $k' = 3k + 3 \cdot \binom{k}{2} + 1$ has the following form.

- (I). For each $i \in [k]$, $V(F) \cap Z_{+\alpha}(i) = \{\alpha_x^i, \alpha_y^i, z_s^i\}$, for some $s \in [p]$.
- (II). For each $1 \leq i < j \leq k$, $V(F) \cap R_{+\alpha}(i, j) = \{\alpha_x^{(i,j)}, \alpha_y^{(i,j)}, r_{t_i, t_j}^{(i,j)}\}$, for some $t, t' \in [p]$.
- (III). $\beta \in V(F)$.

To conclude the proof, we argue that any such induced forest F includes an independent set of size $k + \binom{k}{2}$ of the form (9.11). In particular, we use the following claim to establish the correctness of the reduction.

Claim 9.11.6. *Let F be an induced forest in G' on k' vertices, $1 \leq i < j \leq k$ and $s_i, s_j, t_i, t_j \in [p]$. If $z_{s_i}^i, r_{t_i, t_j}^{(i,j)}, z_{s_j}^j \in V(F)$, then $s_i = t_i$ and $s_j = t_j$.*

Proof. Suppose not and assume wlog. that $s_i \neq t_i$. Recall that by (III), we can assume that $\beta \in V(F)$, and by construction, β is adjacent to all vertices in $Z(i)$ and $R(i, j)$, so in particular β is adjacent to $z_{s_i}^i$ and $r_{t_i, t_j}^{(i,j)}$. However, by Observation 9.3 and the assumption that $s_i \neq t_i$, we have that $z_{s_i}^i r_{t_i, t_j}^{(i,j)} \in E(G')$, hence $\{\beta, z_{s_i}^i, r_{t_i, t_j}^{(i,j)}\}$ induces a cycle in F , a contradiction. \square

Since by (I) and (II), any induced forest on k' vertices contains precisely one vertex from each $Z(i)$ (for $i \in [k]$) and $R(i, j)$ (for $1 \leq i < j \leq k$), we can conclude together with Claim 9.11.6 that $V(F)$ contains an independent set

$$\{z_{s_1}^1, \dots, z_{s_k}^k\} \cup \{r_{s_i, s_j}^{(i,j)} \mid 1 \leq i < j \leq k\}$$

which by Observation 9.3 implies that G has a clique on vertex set $\{v_{s_1}^1, \dots, v_{s_k}^k\}$ which proves the correctness of the reduction.

Finally, since the size of G' is polynomial in the size of G , $k' = 3k + 3 \cdot \binom{k}{2} + 1$, and $|H| = k(3k - 1)$ (see (9.10)), we can conclude that MAXIMUM INDUCED FOREST on H -graphs is W[1]-hard parameterized by $k + |H|$. \square

As in both directions of the correctness proof in the above reduction, the solution to MAXIMUM INDUCED FOREST is connected, it shows hardness for the MAXIMUM INDUCED TREE problem in the same parameterization as well. Furthermore, since a graph on n vertices has an induced forest of size k if and only if it has a feedback vertex set of size $n - k$, we have the following consequence of Theorem 9.11.

Corollary 9.12. MAXIMUM INDUCED TREE on H -graphs is W[1]-hard parameterized by $k + ||H||$, where k denotes the solution size, and FEEDBACK VERTEX SET on H -graphs is W[1]-hard parameterized by $||H||$, and in both cases the hardness holds even if an H -representation of the input graph is given.

By Theorem 4.27, we know that the linear mim-width of an H -graph is at most $2 \cdot ||H||$ and a linear branch decomposition achieving this bound can be computed in polynomial time from a given H -representation of the graph in question. Theorem 9.11 and Corollary 9.12 therefore imply the following.

Corollary 9.13. MAXIMUM INDUCED FOREST and MAXIMUM INDUCED TREE are W[1]-hard parameterized by $k + w$, and FEEDBACK VERTEX SET is W[1]-hard parameterized by w , where k denotes the solution size and w the linear mim-width of the input graph. In both cases, the hardness holds even if a linear branch decomposition of mim-width w is given.

9.5 Discussion and Open Problems

The NP-hardness of HAMILTONIAN CYCLE on graphs of linear mim-width 1 raises new questions about the graph class LINEAR MIM-WIDTH 1. On all graph classes where the linear mim-width was known to be bounded by 1, HAMILTONIAN CYCLE is polynomial-time solvable, so our hardness result hints at this class being much larger than expected. This deems it worthwhile to investigate graphs of linear mim-width 1 more closely.

Open Problem 9.1. Concisely characterize the graph class LINEAR MIM-WIDTH 1.

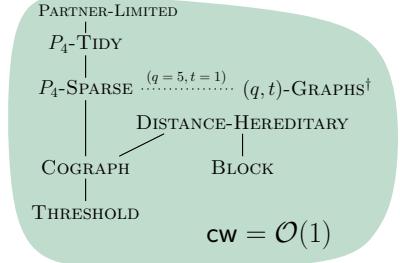
It would be surprising if any of the (σ, ρ) -domination problems parameterized by mim-width turned out to be FPT. Nevertheless, for several such problems, including for instance PERFECT CODE (see Table 9.1), hardness results are not known yet. Bergougnoux and Kanté [21] considered connectivity and acyclicity variants of (σ, ρ) -problems, as well as problems asking for complements of (σ, ρ) sets. Is there a unified lower bound theory for these problems as well?

Open Problem 9.2. Are all connected, acyclic, (co-) (σ, ρ) -problems parameterized by (linear) mim-width W[1]-hard?

Lastly, it would be interesting to obtain tight lower bounds under the ETH. It is known that MULTICOLORED CLIQUE parameterized by k does not have an

$f(k)n^{o(k)}$ time algorithm unless ETH fails; the reductions presented above usually infer a quadratic blow-up in the parameter w (being the mim-width), ruling out only $n^{o(\sqrt{w})}$ time algorithms under ETH.

Open Problem 9.3. Would an $n^{o(w)}$ -time algorithm for some (σ, ρ) -problem parameterized by the (linear) mim-width w of the input graph refute ETH?



[†] $q, t = \mathcal{O}(1); q \geq 4, t \geq 0$, and either
 $q \leq 6$ and $t \leq q-4$, or $q \geq 7$ and $t \leq q-3$

10

Coloring Problems on Clique-Width

Vertex coloring problems are central in algorithmic graph theory, and appear in many variants. In the classical GRAPH COLORING problem, we are given a graph G and an integer k , and the question is whether we can assign each vertex of G one of k colors, such that no pair of adjacent vertices receives the same color. From the perspective of parameterized complexity, GRAPH COLORING parameterized by clique-width is an intriguing, according to the title of the paper [126] an ‘odd’ case. Already in 1994, Wanke [286] (see also [114]) showed that on n -vertex graphs given together with a clique-width w -expression¹, GRAPH COLORING can be solved in XP time, specifically in time $n^{\mathcal{O}(2^w)}$. About 25 years later, Fomin et al. [125] proved that the problem is W[1]-hard in this parameterization, resolving an open problem that has been repeatedly stated in the literature, e.g. [141, 191, 192]. This however does not rule out that the exponential dependence on w in the degree of the polynomial could be avoided; under ETH, the W[1]-hardness proof in [125] only rules out $n^{o(\sqrt[4]{w})}$ -time algorithms. Finally, Fomin et al. [126] proved that not even the exponential dependence of the degree of the polynomial can be avoided, showing that an $n^{2^{o(w)}}$ -time algorithm would refute ETH. This was the first parameterized problem that was known to require an XP-runtime whose degree of the polynomial depends exponentially on the parameter.

On the other hand, as discussed in Section 4.7, several non-trivial graph classes have bounded clique-width, so even the XP-algorithm for GRAPH COLORING gives polynomial-time algorithms on several graph classes at once.

In this chapter, we consider b -COLORING, FALL COLORING, and CLIQUE COLORING parameterized by clique-width, and give XP-algorithms for all of them. Our algorithm for b -COLORING captures most previously known polynomial-time algorithms for the problem when restricted to graph classes and resolves two open problems in the literature, while the algorithm for CLIQUE COLORING is one of the first results regarding structural parameterizations of the problem.

¹The results in [286] are stated in terms of NLC-width, which is linearly related to clique-width.

Complexity of Coloring Problems Parameterized by Clique-Width. We briefly touch on differences in the complexity of vertex coloring problems of graphs when parameterized by clique-width.

We first argue that some generalizations of GRAPH COLORING are NP-complete on graphs of constant clique-width. In the LIST COLORING problem we are given a graph G and for each of its vertices v a list $L(v)$ of colors, and the question is whether G has a proper coloring such that each vertex is assigned a color from its list. This problem is NP-complete on the (not disjoint) union of two complete graphs [173], and such graphs are distance-hereditary, and therefore of clique-width at most 3.

In the related PRECOLORING EXTENSION problem, we are given a graph, some of whose vertices already received a color, and the question is whether this coloring can be extended to a proper coloring of the entire graph. The following standard reduction from LIST COLORING, starting with a graph that is the union of two complete graphs, shows that this variant is NP-complete on graphs of constant clique-width as well. Take the graph G together with the lists $L(\cdot)$, and construct a graph H by adding to G , for each vertex $v \in V(G)$ and each color $c \notin L(v)$, a new vertex x_v^c which is adjacent only to v and assigned color c . It is not difficult to see that this precoloring of H can be extended to the remainder of its vertices if and only if G has a list coloring using the lists $L(\cdot)$. Adding pendant vertices is one of the operations used to construct distance-hereditary graphs, therefore the graph H is distance-hereditary, meaning of clique-width at most 3, as well.

Belmonte et al. [16] recently showed that the GRUNDY COLORING problem, which asks for a linear order of the vertices that maximizes the number of colors used by the greedy coloring heuristic, is NP-complete on graphs of constant clique-width. This nicely contrasts our XP-algorithm for b -COLORING, since both the b -COLORING and the GRUNDY COLORING problems are rooted in the theoretical analysis of graph coloring heuristics.

Decompositions at Hand. All algorithms in this section are given in terms of the equivalent measure *module-width*, and assume that we are given a branch decomposition of bounded module-width w together with the input graph. As seen in Section 3.3, for our purposes, this is equivalent to the assumption that we are given a clique-width w -expression of the input graph. For an overview of algorithms that can be used to construct such expressions (typically approximations), we refer to Section 3.3.2. We assume the reader to be familiar with notions regarding branch decompositions and module-width, in particular the definitions and notation around the *operator* at any node in the branch decomposition. See Section 3.3 for a reminder.

Notation for Colorings. Before we proceed with the presentation of the algorithms, we fix the notation regarding colorings of graphs. Let G be a graph. An ordered partition $\mathcal{C} = (C_1, \dots, C_k)$ of $V(G)$ is called a *coloring* of G (with k colors). (Observe that for $i \in \{1, \dots, k\}$, C_i may be empty.) For all $i \in \{1, \dots, k\}$, we call C_i

a *color class*, and may say that the vertices in C_i have color i . \mathcal{C} is called *proper* if for all $i \in \{1, \dots, k\}$, C_i is an independent set in G .

A proper coloring (C_1, \dots, C_k) is called a *b-coloring*, if for all $i \in \{1, \dots, k\}$, there is a vertex $v_i \in C_i$, called *b-vertex of color i*, such that for all $j \in \{1, \dots, k\} \setminus \{i\}$, $N_G(v_i) \cap C_j \neq \emptyset$.

Given a vertex set $S \subseteq V(G)$ and a coloring \mathcal{C} , we say that S is *monochromatic in \mathcal{C}* if there is a color class $C \in \mathcal{C}$ such that $S \subseteq C$.

The *restriction* of a coloring $\mathcal{C} = (C_1, \dots, C_k)$ to a vertex set $S \subseteq V(G)$, is $\mathcal{C}|_S := (C_1 \cap S, \dots, C_k \cap S)$. In this case we say conversely that \mathcal{C} *extends* $\mathcal{C}|_S$.

10.1 *b*-Coloring

A *b-coloring* with k colors of a graph G is a partition of the vertices of G into k independent sets such that each of them contains a vertex that has a neighbor in all of the remaining ones. The *b-chromatic number* of G , denoted by $\chi_b(G)$, is the maximum integer k such that G admits a *b-coloring* with k colors. This notion was introduced by Irving and Manlove [164] to describe the behavior of the following color-suppressing heuristic for the GRAPH COLORING problem. We start with some proper coloring of the input graph G and try to iteratively suppress one of its colors. That is, for a given color c , we consider each vertex v of color c , and check if there is another color $c' \neq c$ available that does not appear in its neighborhood. If so, we assign vertex v the color c' , observing that the coloring remains proper, and repeat this process for the remaining vertices of color c . If successful, we remove the color c from all vertices of G and decrease the number of colors by one. Once no color can be suppressed by this procedure, the coloring at hand is a *b-coloring* of G , and in the worst case, this heuristic produces a coloring with $\chi_b(G)$ many colors.

Since then, the *b-COLORING* and *b-CHROMATIC NUMBER* problems which given a graph G and an integer k ask whether G has a *b-coloring* with k colors and whether $\chi_b(G) \geq k$, respectively, have received considerable attention in the algorithms and complexity communities.² While these problems have been shown to be NP-complete in the general case [164], as well as on bipartite graphs [196], co-bipartite graphs [47], chordal graphs [155], and line graphs [66], a lot of effort has been put into devising polynomial-time algorithms for these problems in various other classes of graphs. These include trees [164], claw-free block graphs [69], tree-cographs [47], and graphs with few P_4 s, such as cographs and P_4 -sparse graphs [46], P_4 -tidy graphs [283], and $(q, q - 4)$ -graphs for constant q [68]. A common property shared by these graph classes is that they all have bounded *clique-width*, see Section 4.7.³

²We would like to remark that the *b-COLORING* and *b-CHROMATIC NUMBER* problems are not as closely related as the GRAPH COLORING and CHROMATIC NUMBER problems: If a graph G has a *b-coloring* with k colors, then $\chi_b(G) \geq k$, but $\chi_b(G) \geq k$ does not imply the existence of a *b-coloring* with k colors.

³To the best of our knowledge, the only polynomial-time result for graphs of unbounded clique-

In this section we show that b -COLORING (and b -CHROMATIC NUMBER) can be solved in polynomial time on graphs of constant clique-width. Besides unifying the above mentioned polynomial-time cases, this extends the tractability landscape of these problems to larger graph classes, and answers two open problems stated in the literature.

Over a decade ago, Bonomo et al. [46] asked whether their polynomial-time result for P_4 -sparse graphs can be extended to distance-hereditary graphs. Havet et al. [155] answered the question negatively by providing an NP-completeness proof for chordal distance-hereditary graphs. Their proof, however, has a flaw and while it does prove the claimed statement for chordal graphs, it unfortunately fails to do so for distance-hereditary graphs.⁴ Our polynomial-time algorithm for graphs of bounded clique-width in fact provides a positive answer to Bonomo et al.'s question, as distance-hereditary graphs have clique-width at most 3 [148]. In recent years, even subclasses of distance-hereditary graphs have received significant attention, for instance in the work of Campos and Silva [69]: they provide a polynomial-time algorithm for claw-free block graphs, and ask whether this result can be generalized to block graphs. Our algorithm provides a positive answer to this question as well. Moreover, it extends the known algorithm for $(q, q - 4)$ -graphs [68] (for constant q) to all (q, t) -graphs for constants q and t with $q \geq 4$, $t \geq 0$, and either $q \leq 6$ and $t \leq q - 4$, or $q \geq 7$ and $t \leq q - 3$, by a theorem due to Makowsky and Rotics [214].

Our algorithm runs in time $n^{2^{\mathcal{O}(w)}}$, where n denotes the number of vertices of the input graph which is given together with a clique-width w -expression. As consequences of results due to Fomin et al. [125] and Fomin et al. [126], we observe that b -COLORING parameterized by clique-width is W[1]-hard, and that the exponential dependence on w in the degree of the polynomial cannot be avoided unless the Exponential Time Hypothesis (ETH) fails. Concretely, an algorithm running in time $n^{2^{o(w)}}$ would refute ETH.

From the point of view of parameterized complexity, Panolan et al. [242] showed that b -CHROMATIC NUMBER parameterized by the number of colors is W[1]-hard. However, this problem may even be harder, since so far no XP-algorithm is known. Recently, Aboulker et al. [1] showed that the more restrictive b -CHROMATIC CORE problem parameterized by the number of colors (which has a brute-force XP-algorithm, see e.g. [108]) remains W[1]-hard.

It is therefore natural to ask which additional restrictions can be imposed to obtain parameterized tractability results. For instance, answering an open problem posed by Sampaio [261] (see also [269]), it can be observed that b -COLORING on chordal graphs is FPT parameterized by the number of colors [170]. Other restricted cases that have been considered in the literature target specific numbers of colors that depend on the input graph. The DUAL b -COLORING problem, which asks if an

width so far concerns graphs of large girth. In particular, Campos et al. [67] showed that b -CHROMATIC NUMBER is polynomial-time solvable on graphs of girth at least 7.

⁴For details, see [170].

input n -vertex graph has a b -coloring with $n - k$ colors, is FPT parameterized by k [156]. Moreover, deciding if a graph G has a b -coloring with $k = \Delta(G) + 1$ colors, which is an upper bound on $\chi_b(G)$, is FPT parameterized by k [242, 261], while the case $k = \Delta(G)$ is XP and for every fixed $p \geq 1$, the case $k = \Delta(G) - p$ is NP-complete for $k = 3$ [169].

Another novelty aspect of our XP-algorithm parameterized by clique-width is that it is the first result about *structural parameterizations* of the b -COLORING and b -CHROMATIC NUMBER problems. In all previously known polynomial-time cases the algorithms only work if the input graph has some prescribed structure. Our algorithm works on all graphs, albeit with a prohibitively slow runtime on graphs of large clique-width.

The formal definition of the problem we are dealing with is as follows. We work with branch decompositions of bounded *module-width*, rather than clique-width w -expressions, which is equivalent for our purposes by Theorem 3.14 (page 29).

b -COLORING parameterized by $w := \text{mw}(T, \mathcal{L})$

Input: A graph G and one of its rooted branch decompositions (T, \mathcal{L}) , integer k .

Question: Does G have a b -coloring with k colors?

Hardness

Before we proceed, we observe that b -COLORING is W[1]-hard in this parameterization, and that the exponential dependence on w of the degree of the polynomial in the runtime is probably difficult to avoid.

Proposition 10.1. *The b -COLORING problem on graphs on n vertices given together with a linear rooted branch decomposition of module-width w cannot be solved in time $n^{2^{o(w)}}$, unless ETH fails. Moreover, b -COLORING parameterized by w is W[1]-hard.*

Proof. Fomin et al. [126] showed that the GRAPH COLORING problem which given a graph G and an integer k asks for a proper coloring of G with k colors cannot be solved in time $n^{2^{o(w)}}$ unless ETH fails, where w is the module-width of a given linear branch decomposition of the input graph. Using GRAPH COLORING in this setting as a starting point of a reduction, we can add a k -clique to the input graph. The resulting graph has a b -coloring with k colors if and only if the original graph has a proper coloring with k colors (take the vertices in the k -clique as the b -vertices). It is not difficult to see that the given branch decomposition can be extended to include the vertices of the added k -clique without increasing its module-width by too much. W[1]-hardness parameterized by w can be observed using the same argument, even as a consequence of an earlier result [125]. \square

10.1.1 Outline of the Algorithm

Throughout the following, we are given a graph G and one of its rooted branch decompositions (T, \mathcal{L}) of module-width $w = \text{mw}(T, \mathcal{L})$ and we want to find a b -coloring of G with k colors, if it exists. In particular, our algorithm will find a b -coloring \mathcal{C} together with a set of *witness b-vertices*, containing precisely one b -vertex for each color class of \mathcal{C} , if it exists. This will be done via dynamic programming along T , and for each node $t \in V(T)$, the partial solutions associated with t are partial b -colorings of G_t .

Definition 10.2 (Partial b -Coloring). Let G be a graph and $k \in \mathbb{N}$. For an induced subgraph H of G , a *partial b -coloring* of H is a pair (\mathcal{C}, B) of a proper coloring $\mathcal{C} = (C_1, \dots, C_k)$ of H and a subset $B \subseteq V(H)$ such that for all $i \in [k]$, $|C_i \cap B| \leq 1$. We call the vertices in B the *partial b -vertices*.

To obtain an efficient algorithm, we require a compact representation of the partial b -colorings of each subgraph G_t associated with a node $t \in V(T)$. To that end, we introduce the notion of a *t -signature* of a partial b -coloring. Two partial b -colorings with the same t -signature will be interchangeable for the sake of our algorithm, therefore the number of table entries at each node t will be bounded by the number of t -signatures.

Let (\mathcal{C}, B) be a partial b -coloring of G_t . For (\mathcal{C}, B) to be extended to a b -coloring (\mathcal{C}', B') of the entire graph G , we have to ensure that two things happen for each color class $C \in \mathcal{C}$:

- (I) The extension of C in \mathcal{C}' is an independent set in G .
- (II) There is a witness b -vertex in B' for the extension of C in \mathcal{C}' .

The t -signature has to represent a partial b -coloring faithfully enough so that we can keep track of all the ways in which the above two conditions can be satisfied for each of its color classes ‘in the future’. At the same time, its definition has to enable us to significantly compress the information about partial b -colorings of G_t . This happens in the following way. We categorize color classes of partial b -colorings of G_t according to *t -types*. If two color classes C_1, C_2 of a partial b -coloring (\mathcal{C}, B) have the same t -type, then the above two conditions can be satisfied for C_1 and C_2 by extensions of (\mathcal{C}, B) in the exact same ways. This allows us to forget about the ‘names’ of the color classes in a partial b -coloring, but instead to only remember for each t -type how many color classes with that type there are. This is precisely the information that is stored in a t -signature.

Now, if we can bound the number of t -types by some function of the module-width w , say $f(w)$, then the number of t -signatures is upper bounded by $k^{f(w)} \leq n^{f(w)}$. (There are at most k colors, so in particular there are at most k colors with a given t -type.) This translates directly to an upper bound on the number of table entries

in the dynamic programming algorithm, which, up to some constants in the degree of the polynomial, bounds the runtime of the resulting algorithm.

Let us discuss the information that goes into the definition of a t -type. Let C be a color class in a partial b -coloring (\mathcal{C}, B) of G_t . To keep track of which vertices from \overline{V}_t can be added to C without introducing a coloring conflict, it suffices to store which equivalence classes of \sim_t have vertices in C ,⁵ since all vertices in a given equivalence class have the same neighbors in \overline{V}_t . This way we can ensure that condition I is satisfied.

To verify if condition II is satisfied we have to store some information about the partial b -vertices. Naturally, we record whether or not B contains a partial b -vertex of C , but we need to store more information. Suppose that B contains the partial b -vertex v of C . In a straightforward approach, we would simply keep track of the color classes that already appear in the neighborhood of v . This way we could easily decide at which point during the execution of the algorithm, a partial b -vertex turns into a b -vertex. However, this results in prohibitively large table entries, since there are 2^{k-1} subsets of colors that we would have to consider, which for our purpose is no better than 2^n .

We overcome this issue with the following symmetry breaking trick: We do *not* record which color classes the partial b -vertex of C already sees/still needs to see. Instead, we record for which equivalence classes $Q \in V_t/\sim_t$ we need to add a *future neighbor* of Q , i.e. a vertex from $N(Q) \cap \overline{V}_t$, to C , such that the partial b -vertex from *some other color* C' sees color C in its neighborhood. More slowly, suppose that some equivalence class $Q \in V_t/\sim_t$ contains the partial b -vertex $w \in B$ of another color class $C' \neq C$, such that w has no neighbor of color C in V_t . For w to become a b -vertex of its color, color class C must be extended with a neighbor of w in the future, i.e. in \overline{V}_t . The neighborhood of w in \overline{V}_t is precisely $N_G(Q) \cap \overline{V}_t$, therefore we can concisely model this situation as color class C requiring to contain a vertex among the future neighbors of Q . In this situation, we say that

color class C has demand to the future neighbors of Q .

The t -type records for each equivalence class Q of \sim_t , if a color class contains vertices of Q , or if it has demand to the future of Q , or none of the two. Note that if a color class both contains a vertex from Q and has demand to the future of Q , we already know that we can disregard the corresponding partial b -coloring: In the corresponding color class, we cannot add any future neighbors of Q without creating a coloring conflict, and if we do not add a future neighbor of Q , then there is some color class whose partial b -vertex will never become a b -vertex.

Now, if we have a partial b -coloring in which every color class has a partial b -vertex, and all demands have been fulfilled, meaning that there is no color class that has demand to the future of some equivalence class of \sim_t , then we know that we

⁵This is similar to the algorithm of Wanke for GRAPH COLORING on graphs of bounded NLC-width [286].

actually have a b -coloring. Moreover, the number of t -types is $2^{\mathcal{O}(w)}$, so the resulting algorithm runs in time $n^{2^{\mathcal{O}(w)}}$ (see above).

10.1.2 t -Types and t -Signatures

In this section we introduce the basic concepts that we alluded to in the above description, namely the notion of a *t -type* and of a *t -signature*, where t is some node in the given branch decomposition. A *t -type* is meant to capture the necessary information of a color class in a partial b -coloring of G_t . However, we cannot give the definition of a *t -type* as a property of a vertex set alone: a color class C may have demand to the future of an equivalence class, which is because there is a partial b -vertex of *another* color $C' \neq C$ that has no neighbor of color C yet. Therefore, we first give the definition of a *t -type* abstractly, i.e. absent of any partial b -coloring or color class, and then define what it means for a color class to be of a certain *t -type* *within a partial b -coloring*.

The *t -type* is a pair of a bit that is meant to tell us whether or not a coloring contains a partial b -vertex of that color, and a map that tells us for each equivalence class, whether there is a vertex of the color in the equivalence class, or if the color has demand to the future neighbors of the equivalence class, or none of the two.

Definition 10.3 (t -Type). Let G be a graph with rooted branch decomposition (T, \mathcal{L}) and let $t \in V(T)$. A *t -type* is a pair (ϕ, ξ) of a map $\phi: Q_t / \sim_t \rightarrow \{\text{none, cont, dem}\}$ and a bit $\xi \in \{0, 1\}$. We denote the set of all *t -types* by types_t .

Before we proceed, we observe an upper bound on the number of *t -types*. For the component ξ , we clearly only have two choices, and for each equivalence class Q of \sim_t , the entry $\phi(Q)$ takes one of three values.

Observation 10.4. Let (T, \mathcal{L}) be a rooted branch decomposition of module-width $w = \text{mw}(T, \mathcal{L})$. For each $t \in V(T)$, $|\text{types}_t| = 2 \cdot 3^{|V_t / \sim_t|} \leq 2 \cdot 3^w$.

We now define what it means for a color class to be of a certain *t -type* within a partial b -coloring of G_t . This is basically a formalization of the above discussion, but it holds one additional aspect that is of importance of the algorithm and the arguments to follow. We discuss this after the following definition, which is illustrated in Figure 10.1.

Definition 10.5. Let G be a graph with rooted branch decomposition (T, \mathcal{L}) and let $t \in V(T)$. Let (\mathcal{C}, B) be a partial b -coloring of G_t , let $C \in \mathcal{C}$ be a color class, and let $(\phi, \xi) \in \text{types}_t$ be a *t -type*. We say that C *has t -type τ in (\mathcal{C}, B)* if

- (i) $\xi = |C \cap B|$ and
- (ii) for each $Q \in V_t / \sim_t$,

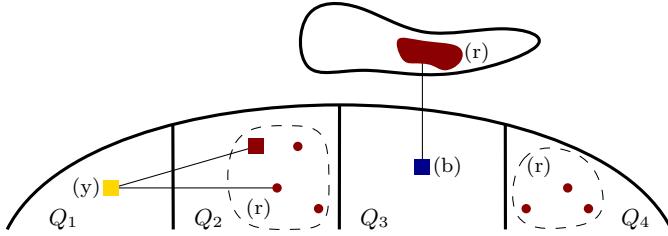


Figure 10.1: Illustration of the definition of a color class being of a certain *t-type* inside a partial *b*-coloring of G_t . The large square vertices are partial *b*-vertices for their color. The type of the red (*r*) color in the coloring is as follows. Since it has a *b*-vertex (the one in Q_2), we have that $\xi = 1$. Since Q_2 and Q_4 have red vertices, $\phi(Q_2) = \phi(Q_4) = \text{cont}$. Q_1 and Q_3 do not have red vertices. Q_1 contains the *b*-vertex of color yellow (*y*), but this vertex already has a red neighbor. Therefore, $\phi(Q_1) = \text{none}$. Finally, Q_3 has the *b*-vertex of color blue (*b*), and this vertex does not have a red neighbor yet. Therefore, there has to be a red vertex among the future neighbors of Q_3 . Hence, $\phi(Q_3) = \text{dem}$.

- (a) if $Q \cap C \neq \emptyset$, and there is no $v \in (B \setminus C) \cap Q$ such that $N(v) \cap C = \emptyset$, then $\phi(Q) = \text{cont}$,
- (b) if $Q \cap C = \emptyset$ and there exists some $v \in (B \setminus C) \cap Q$ such that $N(v) \cap C = \emptyset$, then $\phi(Q) = \text{dem}$, and
- (c) if $Q \cap C = \emptyset$, and there is no $v \in (B \setminus C) \cap Q$ such that $N(v) \cap C = \emptyset$, then $\phi(Q) = \text{none}$.

The reader may have observed that (ii) does not cover all the possibilities. The situation that is not covered is when $Q \cap C \neq \emptyset$ and there is some $v \in (B \setminus C) \cap Q$ such that $N(v) \cap C \neq \emptyset$. A priori, we can of course not exclude this as a possibility, but there is a simple reason that partial *b*-colorings that contain a color class in which this situation arises can be disregarded: For the vertex v to become a *b*-vertex for its color, we have to add a future neighbor of Q to C ; but since Q already contains a vertex from C this means that the resulting set is not independent anymore.

We turn to the definition of a *t*-signature which again is first given in abstract terms.

Definition 10.6 (*t*-Signature). Let G be a graph with rooted branch decomposition (T, \mathcal{L}) , and let $t \in V(T)$. A *t-signature* is a map $\text{sig}_t : \text{types}_t \rightarrow \{0, 1, \dots, k\}$ such that $\sum_{\tau \in \text{types}_t} \text{sig}_t(\tau) = k$.

The following bound on the number of *t*-signatures immediately follows from Observation 10.4: for each *t*-type, the function takes one of $k + 1 \leq n + 1$ values.

Observation 10.7. Let G be a graph on n vertices and (T, \mathcal{L}) be one of its branch decompositions of module-width $w = \text{mw}(T, \mathcal{L})$. For each $t \in V(T)$, there are at most $n^{2^{\mathcal{O}(w)}}$ many *t*-signatures.

A *t-signature representing* a partial *b-coloring* (\mathcal{C}, B) of G_t (with k colors) means that the function counts correctly for each *t-type* how many color classes in \mathcal{C} are of that *t-type* in (\mathcal{C}, B) .

Definition 10.8. Let G be a graph with rooted branch decomposition (T, \mathcal{L}) , and let $t \in V(T)$. Let furthermore sig_t be a *t-signature* and (\mathcal{C}, B) a partial *b-coloring* in G_t . We say that sig_t *represents* (\mathcal{C}, B) if for each *t-type* $\tau \in \text{types}_t$, there are precisely $\text{sig}_t(\tau)$ color classes in (\mathcal{C}, B) that have *t-type* τ in (\mathcal{C}, B) .

If for some partial *b-coloring* (\mathcal{C}, B) of G_t , there is no *t-signature* that represents (\mathcal{C}, B) , then we say that (\mathcal{C}, B) is *not representable*.

We would like to remark again that not all partial *b-colorings* of G_t can be represented by a *t-signature*, since there is a case that a color class cannot be described by a *t-type*. In this case the partial *b-coloring* is not representable. Conversely, we can make the following observation about representable partial *b-colorings* which is useful in several proofs and sometimes used without explicit reference.

Observation 10.9. Let G be a graph with rooted branch decomposition (T, \mathcal{L}) , and let $t \in V(T)$. Let (\mathcal{C}, B) be a representable partial *b-coloring* of G_t , and let $C \in \mathcal{C}$ be a color class whose *t-type* in (\mathcal{C}, B) is (ϕ, ξ) . If for some equivalence class $Q \in V_t/\sim_t$, $Q \cap C \neq \emptyset$, then $\phi(Q) = \text{cont}$.

10.1.3 Compatibility

Let $t \in V(T) \setminus L(T)$ be an internal node of the given rooted branch decomposition, let r and s be its children, and let (H_t, η_r, η_s) be the operator of t .⁶ In our algorithm, we want to combine information about partial *b-colorings* of G_r and G_s to obtain information about partial *b-colorings* of G_t . We will try to obtain a color class of a partial *b-coloring* of G_t by taking the union of a color class C_r of a partial *b-coloring* of G_r and a color class C_s of a partial *b-coloring* of G_s .

However, in some cases this is not possible. For instance, when C_r contains vertices from some equivalence class $Q_r \in V_r/\sim_r$ and C_s contains vertices from some equivalence class $Q_s \in V_s/\sim_s$, and in the graph H_t of the operator of t , we have that $Q_r Q_s \in E(H_t)$. Then, in G_t all edges between the set Q_r and Q_s are present which means that $C_r \cup C_s$ is not an independent set in G_t .

Another condition is necessary to ensure that several demands that *have to be* met at node t are indeed met. Let $C_t = C_r \cup C_s$ and suppose there is an equivalence class $Q_t \in V_t/\sim_t$ that contains a vertex of C_t . Suppose furthermore that there is another equivalence class $Q_r \in V_r/\sim_r$ such that C_r has demand to the future neighbors of Q_r . Then, this demand must be fulfilled by a neighbor of C_r in $V_s \cap Q_t$, for otherwise, the equivalence class Q_t of \sim_t whose bubble contains Q_r both contains vertices of C_t and C_t has demand to the future neighbors of Q_t . The resulting partial *b-coloring* would not be representable.

⁶For a reminder of these concepts, see Section 3.3.

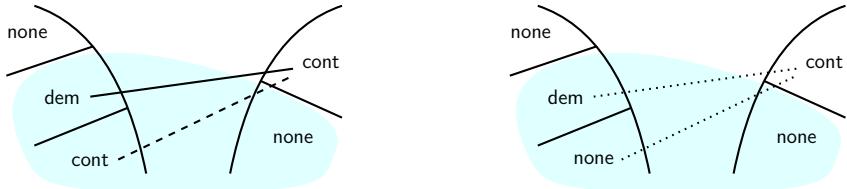


Figure 10.2: Illustration of Definition 10.10. The shaded area shows a bubble and the labels on the equivalence classes correspond to type labelings. For the left hand side, note that between a pair of classes that are both labeled ‘cont’, there can be no edge in the operator. Moreover, since the bubble contains a class labeled cont and one labeled dem, the demand of the latter has to be fulfilled at this node, i.e. there has to be an edge from this class to a ‘cont’-class. The right side shows the situation when the ‘cont’-class in the bubble is changed to ‘none’, in which case the dotted edges may or may not be present in the operator.

The following definition formalizes this discussion and projects it down to the ‘type level’; we illustrate this notion in Figure 10.2.

Definition 10.10 (Compatible types). Let G be a graph with rooted branch decomposition (T, \mathcal{L}) . Let furthermore $t \in V(T) \setminus L(T)$ with children r and s , and let (H_t, η_r, η_s) be the operator of t . Let $(\phi_r, \xi_r) \in \text{types}_r$ and $(\phi_s, \xi_s) \in \text{types}_s$. We say that (ϕ_r, ξ_r) and (ϕ_s, ξ_s) are *compatible* if the following conditions hold.

- (i) $\xi_r + \xi_s \leq 1$.
- (ii) There is no pair $Q_r \in V_r/\sim_r$, $Q_s \in V_s/\sim_s$ such that $Q_r Q_s \in E(H_t)$ and $\phi_r(Q_r) = \phi_s(Q_s) = \text{cont}$.
- (iii) For each $Q \in V_t/\sim_t$ such that there exists a $p \in \{r, s\}$ and a $Q_p \in \eta_p^{-1}(Q)$ with $\phi_p(Q_p) = \text{cont}$, the following holds.
 - (a) For all $Q_r \in \eta_r^{-1}(Q)$ with $\phi_r(Q_r) = \text{dem}$, there is a $Q_s \in V_s/\sim_s$ with $\phi_s(Q_s) = \text{cont}$ and $Q_r Q_s \in E(H_t)$.
 - (b) Similarly, for all $Q_s \in \eta_s^{-1}(Q)$ with $\phi_s(Q_s) = \text{dem}$, there is a $Q_r \in V_r/\sim_r$ with $\phi_r(Q_r) = \text{cont}$ and $Q_s Q_r \in E(H_t)$.

Given a pair of a color class C_r of a partial *b*-coloring of G_r and a color class C_s of a partial *b*-coloring of G_s whose types in the respective colorings are compatible, $C_r \cup C_s$, considered as a color class in a partial *b*-coloring of G_t , has a fixed type. We prove this later in the lemmas that attest the correctness of the algorithm, but we already describe the construction of this type here, mainly since the notion of compatibility of signatures that we give below, requires this ‘merge type’.

Definition 10.11 (Merge Type). Let G be a graph with rooted branch decomposition (T, \mathcal{L}) . Let furthermore $t \in V(T) \setminus L(T)$ with children r and s , and let

(H_t, η_r, η_s) be the operator of t . Let $\rho = (\phi_r, \xi_r) \in \text{types}_r$ and $\sigma = (\phi_s, \xi_s) \in \text{types}_s$ be a pair of compatible types. The *merge type* of ρ and σ , denoted by $\mu(\rho, \sigma)$, is the following t -type (ϕ_t, ξ_t) .

- (i) $\xi_t = \xi_r + \xi_s$.
- (ii) For each $Q \in V_t / \sim_t$:
 - (a) If for some $p \in \{r, s\}$, there exists a $Q_p \in \eta_p^{-1}(Q)$ with $\phi_p(Q_p) = \text{cont}$, then $\phi_t(Q) = \text{cont}$.
 - (b) If (ii.a) does not apply and for some $p \in \{r, s\}$ there exists a $Q_p \in \eta_p^{-1}(Q)$ with $\phi_p(Q_p) = \text{dem}$ and for all $Q_p Q_o \in E(H_t)$ we have that $\phi_o(Q_o) \neq \text{cont}$, then $\phi_t(Q) = \text{dem}$.
 - (c) If neither (ii.a) nor (ii.b) applies, then $\phi_t(Q) = \text{none}$.

Towards a notion of compatibility of signatures, we first define a structure we call *merge skeleton*. Given a node $t \in V(T)$ with children r and s , the merge skeleton is an edge-labeled bipartite graph whose vertices are the r -types and the s -types, with the merge type of a compatible pair of types $\rho \in \text{types}_r$, $\sigma \in \text{types}_s$ written on the edge $\rho\sigma$. Such an edge is meant to represent the fact that taking the union of a color class C_r that has r -type ρ in a partial b -coloring of G_r with a color class C_s that has s -type σ in a partial b -coloring of G_s results in a color class of t -type $\mu(\rho, \sigma)$ in a partial b -coloring of G_t .

Definition 10.12 (Merge skeleton). Let G be a graph and (T, \mathcal{L}) one of its rooted branch decompositions. Let $t \in V(T) \setminus L(T)$ with children r and s . The *merge skeleton* of r and s is an edge-labeled bipartite graph $(\mathfrak{J}, \mathfrak{m})$ where

- $V(\mathfrak{J}) = \text{types}_r \cup \text{types}_s$,
- for all $\rho \in \text{types}_r$, $\sigma \in \text{types}_s$, $\rho\sigma \in E(\mathfrak{J})$ if and only if ρ and σ are compatible, and
- $\mathfrak{m}: E(\mathfrak{J}) \rightarrow \text{types}_t$ is such that for all $\rho\sigma \in E(\mathfrak{J})$, $\mathfrak{m}(\rho\sigma)$ is the merge type of ρ and σ .

Now, any pair of an r -signature sig_r and an s -signature sig_s can ‘flesh out’ the merge skeleton $(\mathfrak{J}, \mathfrak{m})$ of r and s , in the following sense. We can consider the union of sig_r and sig_s as a map labeling the vertices of \mathfrak{J} . Then, an edge-labeling \mathfrak{n} of \mathfrak{J} with integers from $\{0, 1, \dots, k\}$, such that for each vertex of \mathfrak{J} , the sum over its incident edges e of $\mathfrak{n}(e)$ is equal to its vertex label, produces a t -signature sig_t . We can read off how many color classes of each type there are from the edge labeling \mathfrak{n} . In fact, each t -signature can be produced in such a way, as we prove below.

Definition 10.13 (Compatible signatures). Let (T, \mathcal{L}) be a rooted branch decomposition. Let furthermore $t \in V(T) \setminus L(T)$ with children r and s . Let sig_t be a t -signature, let sig_r be an r -signature and sig_s be an s -signature. We say that $(\text{sig}_t, \text{sig}_r, \text{sig}_s)$ is *compatible* if there is a triple $(\mathfrak{J}, \mathfrak{m}, \mathfrak{n})$ such that $(\mathfrak{J}, \mathfrak{m})$ is the merge skeleton of r and s , and $\mathfrak{n}: E(\mathfrak{J}) \rightarrow \{0, 1, \dots, k\}$ is a map with the following properties.

- (i) For all $p \in \{r, s\}$ and all $\pi \in \text{types}_p$, $\sum_{e \in E(\mathfrak{J}): \pi \in e} \mathfrak{n}(e) = \text{sig}_p(\pi)$.
- (ii) For all $\tau \in \text{types}_t$, $\sum_{e \in E(\mathfrak{J}): \mathfrak{m}(e) = \tau} \mathfrak{n}(e) = \text{sig}_t(\tau)$.

We first show that we can test efficiently whether a triple of signatures is compatible.

Lemma 10.14. *Let G be a graph on n vertices and let (T, \mathcal{L}) be one of its rooted branch decomposition of module-width $w = \text{mw}(T, \mathcal{L})$. Let $t \in V(T) \setminus L(T)$ with children r and s . Let sig_t be a t -signature, sig_r be an r -signature, and sig_s be an s -signature. One can decide in time $n^{2^{\mathcal{O}(w)}}$ whether or not $(\text{sig}_t, \text{sig}_r, \text{sig}_s)$ is compatible.*

Proof. We first observe that the merge skeleton can be constructed in $2^{\mathcal{O}(w)}$ time, where $w = (T, \mathcal{L})$: It is easy to see that given two types $\rho \in \text{types}_r$, $\sigma \in \text{types}_s$, we can decide whether or not ρ and σ are compatible in time $w^{\mathcal{O}(1)}$. Moreover, by Observation 10.4, $|\text{types}_r| \leq 2^{\mathcal{O}(w)}$ and $|\text{types}_s| \leq 2^{\mathcal{O}(w)}$, therefore we have to check for $(2^{\mathcal{O}(w)})^2 = 2^{\mathcal{O}(w)}$ pairs of types if they are compatible, and if so, compute their merge type. (This also implies that $|E(\mathfrak{J})| = 2^{\mathcal{O}(w)}$.) Computing a merge type can be done in time $w^{\mathcal{O}(1)}$ as well, simply by following the construction given in Definition 10.11.

We brute-force all candidates for the labeling \mathfrak{n} . Given such a candidate, we can verify in time $2^{\mathcal{O}(w)}$ if it satisfies parts (i) and (ii) of the definition of compatible signatures. Since $|E(\mathfrak{J})| = 2^{\mathcal{O}(w)}$, a trivial upper bound on the number of such candidate labelings is $n^{2^{\mathcal{O}(w)}}$ and therefore the claimed bound follows. \square

10.1.4 Merging and Splitting Partial *b*-Colorings

In this section we show that the notions introduced above work as desired, and the technical lemmas we prove here will be the cornerstone of the correctness proof of the resulting algorithm that we give later.

Bottom to Top

Lemma 10.15. *Let G be a graph with rooted branch decomposition (T, \mathcal{L}) and let $t \in V(T) \setminus L(T)$ be an internal node with children r and s . Let sig_r be an r -signature, sig_s be an s -signature, and sig_t be a t -signature such that:*

- For all $p \in \{r, s\}$, there is a partial *b*-coloring (C_p, B_p) in G_p that is represented by sig_p , and

- $(\text{sig}_t, \text{sig}_r, \text{sig}_s)$ is compatible.

Then, there is a partial b -coloring (\mathcal{C}_t, B_t) of G_t that is represented by sig_t .

Proof. Let $(\mathfrak{J}, \mathfrak{m}, \mathfrak{n})$ be the structure witnessing that $(\text{sig}_t, \text{sig}_r, \text{sig}_s)$ is compatible. We use Algorithm 5 to create the pair (\mathcal{C}_t, B_t) . We first show that (\mathcal{C}_t, B_t) is indeed a partial b -coloring of G_t , and then later that sig_t represents (\mathcal{C}_t, B_t) .

Input : (\mathcal{C}_r, B_r) , (\mathcal{C}_s, B_s) , \mathfrak{J} , and \mathfrak{n} as above
Output: (\mathcal{C}_t, B_t) , where \mathcal{C}_t is a partition of V_t and $B_t \subseteq V_t$.

```

1  $\mathcal{C}'_r \leftarrow \mathcal{C}_r$ ,  $\mathcal{C}'_s \leftarrow \mathcal{C}_s$ ,  $\mathcal{C}_t \leftarrow \emptyset$ ;
2 foreach  $\rho \in \text{types}_r$ ,  $\sigma \in \text{types}_s$  with  $\rho\sigma \in E(\mathfrak{J})$  do
3   Let  $x \leftarrow \mathfrak{n}(\rho\sigma)$ ;
4   for  $i = 1, \dots, x$  do
5     Let  $C_r \in \mathcal{C}'_r$  be of  $r$ -type  $\rho$  and  $C_s \in \mathcal{C}'_s$  be of  $s$ -type  $\sigma$ ;
6      $\mathcal{C}_t \leftarrow \mathcal{C}_t \cup \{C_r \cup C_s\}$ ;
7      $\mathcal{C}'_r \leftarrow \mathcal{C}'_r \setminus \{C_r\}$ ,  $\mathcal{C}'_s \leftarrow \mathcal{C}'_s \setminus \{C_s\}$ ;
8 return  $(\mathcal{C}_t, B_r \cup B_s)$ ;

```

Algorithm 5: Merging (\mathcal{C}_r, B_r) and (\mathcal{C}_s, B_s) according to \mathfrak{J} and \mathfrak{n} .

Claim 10.15.1. (\mathcal{C}_t, B_t) as constructed above is a partial b -coloring of G_t with k colors.

Proof. Since \mathcal{C}_r is a partition of V_r and \mathcal{C}_s is a partition of V_s , and each part of \mathcal{C}_r and \mathcal{C}_s is used precisely once to obtain a part of \mathcal{C}_t in Algorithm 5, it is clear by Definition 10.13(i), that \mathcal{C}_t is a partition of V_t . Together with Definition 10.13(ii) and the definition of a t -signature, this ensures that \mathcal{C}_t has k parts.

We argue that each part $C \in \mathcal{C}_t$ is an independent set. Suppose for a contradiction that C is not an independent set and let $uv \in E(G_t)$ be an edge with $u, v \in C$. By construction, there are $C_r \in \mathcal{C}_r$ and $C_s \in \mathcal{C}_s$ such that $C = C_r \cup C_s$. Moreover, since C_r and C_s are color classes in a coloring, they are independent sets, so we may assume that $u \in C_r$ and $v \in C_s$ (up to renaming). For all $p \in \{r, s\}$, let $\tau_p = (\phi_p, \xi_p)$ be the p -type of C_p in (\mathcal{C}_p, B_p) . Let furthermore $Q_r \in V_r/\sim_r$ be the equivalence class of \sim_r containing u and $Q_s \in V_s/\sim_s$ be the equivalence class of \sim_s containing v . This means that $\phi_r(Q_r) = \phi_s(Q_s) = \text{cont}$. For u and v to be adjacent, the edge $Q_r Q_s$ has to be present in H_t . On the other hand, $\tau_r \tau_s$ is an edge of the merge skeleton which implies that τ_r and τ_s are compatible types; in which case Definition 10.10(ii) forbids the presence of this edge in H_t , a contradiction.

We have shown that \mathcal{C}_t is a proper coloring of G_t , it remains to show that for all $C \in \mathcal{C}_t$, $|C \cap B_t| \leq 1$. Suppose for a contradiction that for some $C \in \mathcal{C}_t$, $|C \cap B_t| > 1$, and let $C_r \in \mathcal{C}_r$, $C_s \in \mathcal{C}_s$ be such that $C = C_r \cup C_s$, as per Algorithm 5. Since for all $p \in \{r, s\}$, (\mathcal{C}_p, B_p) is a partial b -coloring of G_p , we have that $|C_p \cap B_p| \leq 1$, and

clearly $C_r \cap B_s = C_s \cap B_r = \emptyset$. This means that $|C_r \cap B_r| = |C_s \cap B_s| = 1$; and in the r -type (ϕ_r, ξ_r) of C_r in (\mathcal{C}_r, B_r) and the s -type (ϕ_s, ξ_s) of C_s in (\mathcal{C}_s, B_s) , $\xi_r = \xi_s = 1$. But again, (ϕ_r, ξ_r) and (ϕ_s, ξ_s) are compatible, so by Definition 10.10(i), $\xi_r + \xi_s \leq 1$, a contradiction. \square

To prove the lemma, it remains to show that the t -signature sig_t represents (\mathcal{C}_t, B_t) . This is shown via the following claim, with Definition 10.13 ensuring that the numbers work out.

Claim 10.15.2. *Let $C_r \in \mathcal{C}_r$ and $C_s \in \mathcal{C}_s$, and let $\tau_r = (\phi_r, \xi_r)$ be the r -type of C_r in (\mathcal{C}_r, B_r) , and let $\tau_s = (\phi_s, \xi_s)$ be the s -type of C_s in (\mathcal{C}_s, B_s) , such that $C_t = C_r \cup C_s$ is a color class in (\mathcal{C}_t, B_t) . Then, the t -type of C_t in (\mathcal{C}_t, B_t) is $\mu(\tau_r, \tau_s)$.*

Proof. First observe that if $C_t = C_r \cup C_s$ is a color class in (\mathcal{C}_t, B_t) , then τ_r and τ_s are compatible by construction. Let $\tau_t = (\phi_t, \xi_t) = \mu(\tau_r, \tau_s)$. We have to argue that the t -type of C_t in (\mathcal{C}_t, B_t) is indeed (ϕ_t, ξ_t) .

We first show that $|C_t \cap B_t| = \xi_t = \xi_r + \xi_s$. We have that $\xi_t = 1$ if and only if either $\xi_r = 1$ or $\xi_s = 1$ if and only if either $|C_r \cap B_r| = 1$ or $|C_s \cap B_s| = 1$ if and only if $|C_t \cap B_t| = 1$.

Now let $Q \in V_t/\sim_t$. Suppose that $\phi_t(Q) = \text{cont}$; we have to argue that $C_t \cap Q \neq \emptyset$ and that there is no vertex $v \in (B_t \setminus C_t) \cap Q$ with $N(v) \cap C_t = \emptyset$. By the definition of the merge type, there is some $p \in \{r, s\}$ such that there is a $Q_p \in V_p/\sim_p$ with $\eta_p(Q_p) = Q$ and $\phi_p(Q_p) = \text{cont}$. Therefore, $C_p \cap Q_p \neq \emptyset$ which implies that $C_t \cap Q \neq \emptyset$. Now suppose that there is some vertex $v \in (B_t \setminus C_t) \cap Q$ with $N(v) \cap C_t = \emptyset$. This means that there is some $p \in \{r, s\}$ and some $Q_p \in \eta_p^{-1}(Q)$ such that $v \in Q_p$, and $N(v) \cap C_p = \emptyset$. Since C_p has a p -type in (\mathcal{C}_p, B_p) , this means that $C_p \cap Q_p = \emptyset$ and therefore $\phi_p(Q_p) = \text{dem}$. Assume wlog that $p = r$. Since (ϕ_r, ξ_r) and (ϕ_s, ξ_s) are compatible, we have by Definition 10.10(iii) that there is some $Q_s \in V_s/\sim$ with $\phi_s(Q_s) = \text{cont}$ and $Q_r Q_s \in E(H_t)$. But this implies that v has a neighbor in $C_s \subseteq C_t$.

Now suppose that $\phi_t(Q) = \text{dem}$. Then for any $p \in \{r, s\}$ and $Q_p \in V_p/\sim_p$ with $\eta_p(Q_p) = Q$, $\phi_p(Q_p) \neq \text{cont}$, from which we can conclude that $C_t \cap Q = \emptyset$. Furthermore we may assume (up to renaming) that for some $Q_r \in \eta_r^{-1}(Q)$, $\phi_r(Q_r) = \text{dem}$. This means that there is a vertex $v \in (B_r \setminus C_r) \cap Q_r$ with $N(v) \cap C_r = \emptyset$. Furthermore, we have that for all $Q_r Q_s \in E(H_t)$, $\phi_s(Q_s) \neq \text{cont}$, from which we can conclude that v does not have any neighbor in C_s either. Therefore, v has no neighbor in C_t , as required.

Finally, suppose that $\phi_t(Q) = \text{none}$. Again then there is no $Q_p \in \eta^{-1}(Q)$ such that $\phi_p(Q_p) = \text{cont}$. If for all $p \in \{r, s\}$ and all $Q_p \in \eta_p^{-1}(Q)$, $\phi_p(Q_p) = \text{none}$, then it is clear that $C_t \cap Q = \emptyset$, and that there is no $v \in (B_t \setminus C_t) \cap Q$ with $N(v) \cap C_t = \emptyset$. So suppose (up to renaming) that for some $Q_r \in \eta_r^{-1}(Q)$, $\phi_r(Q_r) = \text{dem}$, implying that there is a vertex $v \in (B_r \setminus C_r) \cap Q_r$ with $N(v) \cap C_r = \emptyset$. Since we did not land in case (ii.b) of the definition of a merge type, there is some $Q_r Q_s \in E(H_t)$ such that $\phi_s(Q_s) = \text{cont}$, which means v has a neighbor in $C_s \subseteq C_t$. Since this holds for

any such Q_r (and Q_s), we can conclude that there is no vertex in $(B_t \setminus C_t) \cap Q$ with $N(v) \cap C_t = \emptyset$. This concludes the proof. \square

This concludes the proof of Lemma 10.15. \square

Top to Bottom

Lemma 10.16. *Let G be a graph with rooted branch decomposition (T, \mathcal{L}) and let $t \in V(T) \setminus L(T)$ be an internal node with children r and s . Let sig_t be a t -signature, and suppose there is a partial b -coloring (\mathcal{C}_t, B_t) of G_t which is represented by sig_t . Then, there exists an r -signature sig_r and an s -signature sig_s such that*

- for all $p \in \{r, s\}$ there is a partial b -coloring (\mathcal{C}_p, B_p) represented by sig_p , and
- $(\text{sig}_t, \text{sig}_r, \text{sig}_s)$ is compatible.

Proof. For all $p \in \{r, s\}$, we let $\mathcal{C}_p := \mathcal{C}_t|_{V_p}$ and $B_p := B_t \cap V_p$. It is clear that (\mathcal{C}_p, B_p) is a partial b -coloring of G_p .

Claim 10.16.1. *For all $p \in \{r, s\}$, (\mathcal{C}_p, B_p) is represented by some p -signature.*

Proof. Suppose the claim is false for $p = r$. Then there is some $C_r \in \mathcal{C}_r$ that has no r -type in (\mathcal{C}_r, B_r) , meaning that for some $Q_r \in V_r/\sim_r$, $Q_r \cap C_r \neq \emptyset$ and there is a vertex $v \in (B_r \setminus C_r) \cap Q_r$ with $N(v) \cap C_r = \emptyset$. By construction, there is a $C_t \in \mathcal{C}_t$ with $C_t = C_r \cup C_s$ for some $C_s \subseteq V_s$. Since (\mathcal{C}_t, B_t) is representable, and $\eta_r(Q_r) \cap C_t \neq \emptyset$, we know that $N(v) \cap C_t \neq \emptyset$ (otherwise, C_t has no t -type in (\mathcal{C}_t, B_t)). Therefore, $N(v) \cap C_s \neq \emptyset$. But since all vertices in Q_r are twins with respect to V_s , and since $C_r \cap Q_r \neq \emptyset$ and $v \in Q_r$, this means that there is an edge between some vertex in C_r and some vertex in C_s , contradicting the fact that C_t is an independent set. \square

By the previous claim, we know that (\mathcal{C}_r, B_r) is represented by some r -signature sig_r and that (\mathcal{C}_s, B_s) is represented by some s -signature sig_s . It remains to show that $(\text{sig}_t, \text{sig}_r, \text{sig}_s)$ is compatible. To be able to argue this, we show that each t -type appears as an edge label of the merge skeleton $(\mathfrak{J}, \mathfrak{m})$ of r and s , in particular that it is the merge type of the r -type and s -type labeling the endpoints of this edge.

Claim 10.16.2. *Let $C_t \in \mathcal{C}_t$ be a color class whose t -type in (\mathcal{C}_t, B_t) is $\tau_t = (\phi_t, \xi_t)$. Let $\tau_r = (\phi_r, \xi_r)$ be the r -type of $C_r := C_t \cap V_r$ in (\mathcal{C}_r, B_r) , and let $\tau_s = (\phi_s, \xi_s)$ be the s -type of $C_s := C_t \cap V_s$ in (\mathcal{C}_s, B_s) . Then, $\tau_t = \mu(\tau_r, \tau_s)$.*

Proof. We first have to show that τ_r and τ_s are compatible. We know that $\xi_t \in \{0, 1\}$ and that $\xi_t = 1$ if and only if $|C_t \cap B_t| = 1$ if and only if either $|C_r \cap B_r| = 1$ or $|C_s \cap B_s| = 1$ if and only if either $\xi_r = 1$ or $\xi_s = 1$, therefore $\xi_r + \xi_s \leq 1$, meaning that condition (i) of the definition of compatibility is satisfied. Since C_t is an independent set, there are no edges between C_r and C_s . This means that for any pair $Q_r \in V_r/\sim_r$,

$Q_s \in V_s/\sim_s$ with $\phi_r(Q_r) = \phi_s(Q_s) = \text{cont}$, $Q_r Q_s \notin E(H_t)$, otherwise there would be an edge between C_r and C_s , so condition (ii) is satisfied as well.

Now suppose that Definition 10.10(iii) is violated. We may assume (up to renaming) that there is some $Q \in V_t/\sim_t$ with the following properties. There is a $Q^* \in \eta_r^{-1}(Q)$ with $\phi_r(Q^*) = \text{cont}$, meaning that $C_r \cap Q^* \neq \emptyset$ and so $C_t \cap Q \neq \emptyset$. Moreover, there is some $Q_r \in \eta_r^{-1}(Q)$ with $\phi_r(Q_r) = \text{dem}$, where for any $Q_r Q_s \in E(H_t)$, $\phi_s(Q_s) \neq \text{cont}$. This means that there is a vertex $v \in (B_r \setminus C_r) \cap Q_r$ such that $N(v) \cap C_r = \emptyset$, and moreover that $N(v) \cap C_s = \emptyset$, implying that $N(v) \cap C_t = \emptyset$. Note that $v \in (B_t \setminus C_t) \cap Q_t$. In other words, we have argued that Q is an equivalence class of \sim_t such that $C_t \cap Q \neq \emptyset$ and there is a vertex $v \in (B_t \setminus C_t) \cap Q$ such that $N(v) \cap C_t = \emptyset$. But this means that the color class C_t cannot have a t -type in (C_t, B_t) , so (C_t, B_t) was not representable, a contradiction.

Now we argue that τ_t , the t -type of C_t in (C_t, B_t) , is indeed the merge type of τ_r and τ_s . We already argued above that $\xi_t = \xi_r + \xi_s$. Now let $Q \in V_t/\sim_t$, and suppose that $\phi_t(Q) = \text{cont}$. This means that $Q \cap C_t \neq \emptyset$. We may assume (up to renaming) that $u \in Q_r \cap C_r$ for some $Q_r \in \eta_r^{-1}(Q)$. Since (C_r, B_r) is representable by Claim 10.16.1 this already implies that $\phi_r(Q_r) = \text{cont}$, therefore $\phi_t(Q)$ is set in accordance with the definition of the merge type.

Now suppose that for some $Q \in V_t/\sim_t$, $\phi_t(Q) = \text{dem}$. Then, $Q \cap C_t = \emptyset$ and there is some $v \in (B_t \setminus C_t) \cap Q$ such that $N(v) \cap C_t = \emptyset$. First, since $Q \cap C_t = \emptyset$, this immediately implies that for all $p \in \{r, s\}$ and all $Q_p \in \eta_p^{-1}(Q)$, $Q_p \cap C_p = \emptyset$ and therefore $\phi_p(Q_p) \neq \text{cont}$. Now for $p \in \{r, s\}$, let Q_p be the equivalence class of \sim_p containing v . We may assume (up to renaming) that $p = r$. Clearly, $\eta_r(Q_r) = Q$, therefore $Q_r \cap C_r = \emptyset$. Moreover, $N(v) \cap C_r = \emptyset$, and we have that $\phi_r(Q_r) = \text{dem}$. Now suppose for a contradiction that for some $Q_r Q_s \in E(H_t)$, $\phi_s(Q_s) = \text{cont}$. This implies that $N(v) \cap C_s \neq \emptyset$, and therefore $N(v) \cap C_t \neq \emptyset$, a contradiction. We have shown that also in this case, $\phi_t(Q)$ is set in accordance with the definition of the merge type.

Finally, suppose that $\phi_t(Q) = \text{none}$. Then, $Q \cap C_t = \emptyset$ and there is no $v \in (B_t \setminus C_t) \cap Q$ with $N(v) \cap C_t = \emptyset$. This immediately implies that for all $p \in \{r, s\}$ and all $Q_p \in \eta_p^{-1}(Q)$, $\phi_p(Q_p) \neq \text{cont}$. Suppose that for some $p \in \{r, s\}$ and some $Q_p \in \eta_p^{-1}(Q)$, $\phi_p(Q_p) = \text{dem}$, and assume (up to renaming) that $p = r$. This means that there is some vertex $u \in (B_r \setminus C_r) \cap Q_r$ with $N(u) \cap C_r = \emptyset$. On the other hand, we know that $N(u) \cap C_t \neq \emptyset$, so u has a neighbor in C_s . This means that there is a $Q_r Q_s \in E(H_t)$ such that $Q_s \cap C_s \neq \emptyset$, meaning that $\phi_s(Q_s) = \text{cont}$. Therefore, $\phi_t(Q)$ is also set in accordance with the definition of the merge type. \square

To finish the proof, we have to construct an edge labeling $\mathbf{n}: E(\mathfrak{J}) \rightarrow \{0, 1, \dots, k\}$ satisfying the conditions of Definition 10.13. The previous claim tells us that we can construct \mathbf{n} in a straightforward way. Initially, set $\mathbf{n}(\tau_t) = 0$ for all $\tau_t \in \text{types}_t$. For each color class $C_t \in \mathcal{C}_t$ whose t -type in (C_t, B_t) is τ_t , we know that the r -type of $C_r := C_t \cap V_r$, say τ_r , and the s -type of $C_s := C_t \cap V_s$, say τ_s , are such that $\tau_t = \mathbf{m}(\tau_r \tau_s)$, i.e. τ_t appears as the label of the edge between τ_r and τ_s in \mathfrak{J} .

We therefore increase the value of $\mathbf{n}(\tau_r \tau_s)$ by one. Once we did this for all color classes of (\mathcal{C}_t, B_t) , the tuple $(\mathfrak{J}, \mathfrak{m}, \mathbf{n})$ satisfies the requirements of Definition 10.13, so $(\mathbf{sig}_t, \mathbf{sig}_r, \mathbf{sig}_s)$ is compatible. \square

10.1.5 The Algorithm

As alluded to above, the algorithm is bottom-up dynamic programming along the given rooted branch decomposition (T, \mathcal{L}) of G . First, we define the table entries stored at each node.

Definition of the table entries. For a node $t \in V(T)$ and a t -signature \mathbf{sig}_t , we let $\mathbf{tab}[t, \mathbf{sig}_t] = 1$ if and only if there exists a partial b -coloring of G_t that is represented by \mathbf{sig}_t . \triangleleft

We now show that if all table entries have been computed correctly, then the solution can be read off the table entries stored at the root \mathfrak{r} of the given rooted branch decomposition. Observe that since $V_{\mathfrak{r}} = V(G)$ and therefore $\overline{V}_{\mathfrak{r}} = \emptyset$, the equivalence relation $\sim_{\mathfrak{r}}$ has one equivalence class, namely $V(G)$.

Lemma 10.17. *Let G be a graph with rooted branch decomposition (T, \mathcal{L}) and let $\mathfrak{r} \in V(T)$ be the root of T . Let ρ be the \mathfrak{r} -type $(\phi_{\mathfrak{r}}, \xi_{\mathfrak{r}})$ with $\xi_{\mathfrak{r}} = 1$ and $\phi_{\mathfrak{r}}(V(G)) = \text{cont}$. Let $\mathbf{sig}_{\mathfrak{r}}$ be the \mathfrak{r} -signature letting $\mathbf{sig}_{\mathfrak{r}}(\rho) = k$. Then, G has a b -coloring with k colors if and only if $\mathbf{tab}[\mathfrak{r}, \mathbf{sig}_{\mathfrak{r}}] = 1$.*

Proof. Suppose that G has a b -coloring (\mathcal{C}, B) with k colors. Then, (\mathcal{C}, B) is also a partial b -coloring; but since all vertices in B are already b -vertices for their color, all demands have been fulfilled. This means that (\mathcal{C}, B) is representable by an \mathfrak{r} -signature, denote this \mathfrak{r} -signature by \mathbf{sig} . We argue that $\mathbf{sig} = \mathbf{sig}_{\mathfrak{r}}$, in particular that all color classes $C \in \mathcal{C}$ are of type $\rho = (\phi_{\mathfrak{r}}, \xi_{\mathfrak{r}})$ in (\mathcal{C}, B) as in the statement of the lemma. Let $C \in \mathcal{C}$ be any color class. Since (\mathcal{C}, B) is a b -coloring, B contains a b -vertex v of C , therefore also $C \neq \emptyset$ which implies that the \mathfrak{r} -type of C is indeed ρ . As this reasoning applies to all k color classes of (\mathcal{C}, B) , we can conclude that $\mathbf{tab}[\mathfrak{r}, \mathbf{sig}_{\mathfrak{r}}] = 1$.

Now suppose for the other direction that $\mathbf{tab}[\mathfrak{r}, \mathbf{sig}_{\mathfrak{r}}] = 1$. Then there is a partial b -coloring (\mathcal{C}, B) of $G_{\mathfrak{r}} = G$ with k colors represented by $\mathbf{sig}_{\mathfrak{r}}$. Since $(\phi_{\mathfrak{r}}, \xi_{\mathfrak{r}})$ is the type of each color class and $\xi_{\mathfrak{r}} = 1$, each color class has a partial b -vertex; since no color class has demand to the future neighbors of $V(G)$ by $\phi_{\mathfrak{r}}$, each partial b -vertex is indeed a b -vertex for its color. Therefore, \mathcal{C} is a b -coloring of G with k colors. \square

We describe how to compute the table entries, starting with the leaves of T .

Leaves of T . Let $t \in V(T)$ be a leaf node of T and let $v \in V(G)$ be the vertex such that $\mathcal{L}(v) = t$. We show how to set the table entries $\mathbf{tab}[t, \cdot]$. The partial b -colorings of $G_t = (\{v\}, \emptyset)$ we have to consider are the following. The vertex v is colored with one of the k colors, and it is either the partial b -vertex for its color or not.

The t -signatures representing these colorings look as follows. Observe that \sim_t has precisely one equivalence class, namely $\{v\}$. In the case that v is *not* the partial b -vertex of its color, we have

- one color of type $(\phi_{v,1}, \xi_{v,1})$ with $\xi_{v,1} = 0$ and $\phi_{v,1}(\{v\}) = \text{cont}$, and
- $k - 1$ colors of type $(\phi_\emptyset, \xi_\emptyset)$ with $\xi_\emptyset = 0$ and $\phi_\emptyset(\{v\}) = \text{none}$.

We denote this signature by sig_1 , i.e. we let $\text{sig}_1((\phi_{v,1}, \xi_{v,1})) = 1$ and $\text{sig}_1((\phi_\emptyset, \xi_\emptyset)) = k - 1$.

In the case that v *is* the partial b -vertex of its color class, then the remaining $k - 1$ color classes have demand to the future neighbors of $\{v\}$, so that v eventually becomes the b -vertex of its color. Therefore we have

- one color of type $(\phi_{v,2}, \xi_{v,2})$ with $\xi_{v,2} = 1$ and $\phi_{v,2}(\{v\}) = \text{cont}$, and
- $k - 1$ colors of type $(\phi_{\text{dem}}, \xi_{\text{dem}})$ with $\xi_{\text{dem}} = 0$ and $\phi_{\text{dem}}(\{v\}) = \text{dem}$.

We denote this signature by sig_2 , i.e. we let $\text{sig}_2((\phi_{v,2}, \xi_{v,2})) = 1$ and $\text{sig}_2((\phi_{\text{dem}}, \xi_{\text{dem}})) = k - 1$. To summarize, for each t -signature sig , we let

$$\text{tab}[t, \text{sig}] := \begin{cases} 1, & \text{if } \text{sig} \in \{\text{sig}_1, \text{sig}_2\} \\ 0, & \text{otherwise} \end{cases} \quad \triangleleft$$

Next, the internal nodes of T .

Internal nodes of T . Now let $t \in V(T) \setminus L(T)$ with children r and s . For each t -signature sig_t , we let $\text{tab}[t, \text{sig}_t] = 1$ if and only if there exists a pair $(\text{sig}_r, \text{sig}_s)$ of an r -signature sig_r and an s -signature sig_s such that

(J1) $\text{tab}[r, \text{sig}_r] = 1$ and $\text{tab}[s, \text{sig}_s] = 1$, and

(J2) $(\text{sig}_t, \text{sig}_r, \text{sig}_s)$ is compatible. \triangleleft

Equipped with the lemmas of the previous sections, we can prove correctness of the above algorithm.

Lemma 10.18. *For each $t \in V(T)$ and t -signature sig_t , the above algorithm computes the table entry $\text{tab}[t, \text{sig}_t]$ correctly.*

Proof. We prove the lemma by induction on the height of t . For the base case, when t is a leaf, it is easily verified. From now on we may assume that $t \in V(T) \setminus L(T)$ with children r and s .

First, suppose that the algorithm set $\text{tab}[t, \text{sig}] = 1$. This means that there is a pair $(\text{sig}_r, \text{sig}_s)$ of an r -signature sig_r and an s -signature sig_s such that $\text{tab}[r, \text{sig}_r] = 1$ and $\text{tab}[s, \text{sig}_s] = 1$ and $(\text{sig}_t, \text{sig}_r, \text{sig}_s)$ is compatible. By induction, we know that there is a partial b -coloring of G_r represented by the r -signature sig_r and a partial

b -coloring of G_s represented by the s -signature sig_s . Then, by Lemma 10.15, there is a partial b -coloring of G_t represented by the t -signature sig_t .

Conversely, suppose that there is a partial b -coloring of G_t represented by the t -signature sig_t . Then, by Lemma 10.16, there is a partial b -coloring of G_r represented by an r -signature sig_r and a partial b -coloring of G_s represented by an s -signature sig_s , such that $(\text{sig}_t, \text{sig}_r, \text{sig}_s)$ is compatible. By induction, the algorithm set $\text{tab}[r, \text{sig}_r] = 1$ and $\text{tab}[s, \text{sig}_s] = 1$, and therefore, by the above description, it set $\text{tab}[t, \text{sig}_t] = 1$. \square

We wrap up. By Lemma 10.18, the algorithm computes all table entries correctly, and by Lemma 10.17, the solution to the instance can be determined upon inspecting the table entries associated with the root of the given branch decomposition. Correctness of the algorithm follows.

Regarding the runtime, we observe the following. Given an n -vertex graph with rooted branch decomposition (T, \mathcal{L}) of module-width $w = \text{mw}(T, \mathcal{L})$, we have that $|V(T)| = \mathcal{O}(n)$. (T is essentially a full binary tree on n leaves, so $|V(T)| = 2n - 1$.) Let $t \in V(T)$. If t is a leaf node, then computing the table entries $\text{tab}[t, \cdot]$ takes constant time. If t is an internal node, then by Observation 10.7, we have to compute $n^{2^{\mathcal{O}(w)}}$ table entries. Assume by induction that the table entries associated with the children of t have been computed. For each t -signature sig_t we have to try for $\left(n^{2^{\mathcal{O}(w)}}\right)^2 = n^{2^{\mathcal{O}(w)}}$ pairs of one signature per child whether or not they form a compatible triple together with sig_t . For each triple, this can be done in time $n^{2^{\mathcal{O}(w)}}$ by Lemma 10.14. Therefore, the overall runtime of the algorithm is $n^{2^{\mathcal{O}(w)}}$.

Theorem 10.19. *There is an algorithm that solves b -COLORING in time $n^{2^{\mathcal{O}(w)}}$, where n denotes the number of vertices of the input graph, and w denotes the module-width of a given rooted branch decomposition of the input graph.*

10.1.6 b -Continuity

For a graph G on n vertices, its b -spectrum is the set

$$\mathcal{S}_b(G) := \{k \in [n] \mid G \text{ has a } b\text{-coloring with } k \text{ colors}\}.$$

A graph G is said to be b -continuous, if $\mathcal{S}_b(G) = \{\chi(G), \chi(G) + 1, \dots, \chi_b(G)\}$. The notion of b -continuity was introduced by Kratochvíl et al. [196]. Barth et al. [14] showed that given a graph G , a vertex coloring with $\chi(G)$ many colors, and a vertex coloring with $\chi_b(G)$ many colors, it still remains NP-complete to decide whether or not G is b -continuous. As a consequence of Theorem 10.19, we can decide in polynomial time for each graph of bounded clique-width whether or not it is b -continuous. Given G , the algorithm first computes $\chi(G)$ in time $n^{2^{\mathcal{O}(w)}}$ using the algorithm for GRAPH COLORING parameterized by clique-width, see e.g. [286]. Then, for each $k \geq \chi(G)$, it checks if G has a b -coloring with k colors in time $n^{2^{\mathcal{O}(w)}}$ using

the algorithm of Theorem 10.19. It therefore computes the b -spectrum of G in time $n^{2^{\mathcal{O}(w)}}$ given which it is trivial to verify if G is b -continuous: the minimum value in $\mathcal{S}_b(G)$ is necessarily the chromatic number of G , and the maximum value in $\mathcal{S}_b(G)$ is the b -chromatic number of G . We only have to verify if all values in between are present in $\mathcal{S}_b(G)$.

Corollary 10.20. *There is an algorithm that decides whether a graph G is b -continuous in time $n^{2^{\mathcal{O}(w)}}$, where n denotes the number of vertices of the input graph, and w denotes the module-width of a given rooted branch decomposition of the input graph.*

It is worth noting that several graph classes of bounded clique-width were shown to be b -continuous. Concretely, Magalhães [212] showed it for distance-hereditary graphs, Bonomo et al. [46] for P_4 -sparse graphs, and Velasques et al. [283] for P_4 -tidy graphs. On the other hand, knowing that all members of a graph class are b -continuous does not help in computing b -colorings. Chordal graphs are b -continuous (see e.g. [116]), and b -COLORING is NP-complete on chordal graphs [155].

10.2 Fall Coloring

A *fall coloring* is a special type of b -coloring where *every* vertex needs to have at least one neighbor in all color classes except its own. In other words, it is a partition of the vertex set of a graph into independent dominating sets. As a standalone notion, fall coloring has been introduced by Dunbar et al. [107]. However, since the corresponding FALL COLORING problem falls in the category of locally checkable vertex partitioning problems, it has been shown in earlier work of Telle and Proskurowski [272] to be FPT parameterized by the number of colors plus the treewidth of the input graph, and by Heggernes and Telle [158] to be NP-complete for fixed number of colors. FALL COLORING remains hard further restricted to bipartite [203, 204, 268], chordal [268], or planar [204] graphs. On the other hand, even with unbounded number of colors, it is known to be solvable in polynomial time on strongly chordal graphs [210, 142], threshold graphs and split graphs [223]. In all of these cases, one simply checks whether the chromatic number of the input graph is equal to its minimum degree plus one. To the best of our knowledge, these are the only known polynomial-time cases.

We adapt our algorithm for b -COLORING on graphs of bounded clique-width to solve FALL COLORING, and therefore show that the latter problem is as well solvable in time $n^{2^{\mathcal{O}(w)}}$, where w denotes the clique-width of a given decomposition of the input graph.

Adaptation of the b -Coloring Algorithm

We now show how to adapt the algorithm of Theorem 10.19 to solve the FALL COLORING problem in time $n^{2^{\mathcal{O}(w)}}$ as well. This adaptation in some sense simplifies the

algorithm for b -COLORING, since we do no have to keep track of whether or not a color class has a b -vertex in a partial coloring; *every* vertex has to be a b -vertex. Now, if we can construct a coloring such that each color class is nonempty, and each vertex is a b -vertex for its color, then clearly we have a fall coloring. With small modification, the mechanics of our algorithm for b -COLORING allow for checking if there is a coloring with this property. The main difference will be in the definition of the type of a color class.

Let (C_1, \dots, C_k) be a proper coloring of G_t for some node t , and C_i and C_j be two distinct color classes. If for some $Q \in V_t/\sim_t$, $C_i \cap Q = \emptyset$, and there is *any* vertex $v_j \in C_j$ such that $N(v_j) \cap C_i = \emptyset$, then C_i has demand to the future neighbors of Q : the vertex v_j needs to become a b -vertex of color j , and since it has no neighbor in color class i so far, one of its future neighbors (equivalently, a future neighbor of equivalence class Q), has to receive color i .

The definition of a *t -fall type* can be obtained from the definition of a t -type by dropping the bit ξ which becomes unnecessary in the context of FALL COLORING.

The definition of a color class being of a certain t -fall type becomes the following.

Definition 10.21 (t -Fall-type). Let G be a graph with rooted branch decomposition (T, \mathcal{L}) , and let $t \in V(T)$. A *t -fall type* is a map $\phi: V_t/\sim_t \rightarrow \{\text{none}, \text{cont}, \text{dem}\}$.

Let $\mathcal{C} = (C_1, \dots, C_k)$ be a proper coloring of G_t , and let ϕ be a t -fall type. For $i \in \{1, \dots, k\}$, we say that C_i has *t -fall type ϕ* in \mathcal{C} if for each $Q \in V_t/\sim_t$,

- (i) if $Q \cap C_i \neq \emptyset$, then $\phi(Q) = \text{cont}$,
- (ii) if $Q \cap C_i = \emptyset$ and there is a $v_j \in C_j \cap Q$ (for some $j \neq i$) with $N_{G_t}(v_j) \cap C_i = \emptyset$, then $\phi(Q) = \text{dem}$, and
- (iii) $\phi(Q) = \text{none}$, otherwise.⁷

We again restrict ourselves to finding (partial) colorings that are *representable*, in the sense that there is no color class that both intersects an equivalence class and has demand to its future neighbors. In complete analogy, we define a *t -signature* as a function counting the number of color classes of each t -fall type.

We say that two fall-types are compatible, if they satisfy parts (ii) and (iii) of Definition 10.10, the definition of compatible types in the case of b -COLORING. Part (i) simply disappears since we do not have to keep track of whether or not a color class contains a partial b -vertex. With this in mind, the technical arguments given in Section 10.1.4 go through.

The definition of the table entries is analogous as well, and by an argument parallel to the proof of Lemma 10.17, we can conclude that this information is sufficient to solve the problem.

⁷Having a closer look at the resulting algorithm, one can see that we can omit this case from the definition of the type of a color class. We leave it here for the sake of a clearer analogy with the b -COLORING algorithm.

We discuss the resulting algorithm. For the leaf nodes, we only have to consider colorings with one color class whose fall-type is $\phi_v(\{v\}) = \text{cont}$ and $k - 1$ color classes whose fall-type is $\phi_{\text{dem}}(\{v\}) = \text{dem}$. This is because in any fall-coloring of G , the vertex v has to be a b -vertex for its color. The computation of the internal nodes remains the same. A correctness proof of the algorithm can now be given in the same way as in the proof of Lemma 10.18, and the discussion of the runtime of the algorithm still goes through. We have the following theorem.

Theorem 10.22. *There is an algorithm that solves FALL COLORING in time $n^{2^{\mathcal{O}(w)}}$, where n denotes the number of vertices of the input graph, and w denotes the module-width of a given rooted branch decomposition of the input graph.*

10.3 Clique Coloring

A *clique coloring* with k colors of a graph G is an assignment of one of k colors to each vertex of G such that there is no monochromatic maximal clique, meaning that the vertices of each maximal clique are colored with at least two colors. This notion was introduced in 1991 by Duffus et al. [106], and it behaves quite differently from the classical notion of a proper coloring, which forbids monochromatic edges. Any proper coloring is a clique coloring, but not vice versa. For instance, a complete graph on n vertices only has a proper coloring with n colors, while it has a clique coloring with two colors. Moreover, proper colorings are closed under taking subgraphs. On the other hand, removing vertices or edges from a graph may introduce new maximal cliques, therefore a clique coloring of a graph is not always a clique coloring of its subgraphs, not even of its induced subgraphs.

Also from a complexity-theoretic perspective, CLIQUE COLORING behaves very differently from GRAPH COLORING. Most notably, while it is easy to decide whether a graph has a proper coloring with two colors, Bacsó et al. [9] showed that it is already coNP-hard to decide if a given coloring with two colors is a clique coloring. Marx [216] later proved CLIQUE COLORING to be Σ_2^p -complete for every fixed number of (at least two) colors.

On the positive side, Cochefert and Kratsch [78] showed that the CLIQUE COLORING problem can be solved in $\mathcal{O}^*(2^n)$ time, and the problem has been shown to be polynomial-time solvable on several graph classes. Mohar and Skrekovski [224] showed that all planar graphs are 3-clique colorable, and Kratochvíl and Tuza gave an algorithm that decides whether a given planar graph is 2-clique colorable [195]. For several graph classes it has been shown that all their members except odd cycles on at least five vertices (which require three colors) are 2-clique colorable [9, 11, 65, 70, 98, 185, 243, 266]. Therefore, on these classes CLIQUE COLORING is polynomial-time solvable. Duffus et al. [106] even conjectured in 1991 that perfect graphs are 3-clique colorable, which was supported by many subclasses of perfect graphs being shown to be 2- or 3-clique colorable [5, 9, 75, 98, 106, 224, 243]. However, in 2016,

Charbit et al. [73] showed that there are perfect graphs whose clique colorings require an unbounded number of colors.

Following a similar strategy as in the algorithm for b -COLORING given earlier in this chapter, we provide an XP-time algorithm for CLIQUE COLORING with clique-width as the parameter. The algorithm runs in time $n^{f(w)}$, where w is the clique-width w of a given clique decomposition of the input n -vertex graph and $f(w) = 2^{2^{\mathcal{O}(w)}}$. The double-exponential dependence on w in the degree of the polynomial stems from the notorious property of clique colorings which we mentioned above; namely, that taking induced subgraphs does not necessarily preserve clique colorings. This results in a large amount of information that needs to be carried along as the algorithm progresses.

The formal definition of the problem we consider now is as follows.

CLIQUE COLORING parameterized by $w := \text{mw}(T, \mathcal{L})$

Input: A graph G and one of its rooted branch decompositions (T, \mathcal{L}) , an integer k .

Question: Does G have a clique coloring with k colors?

10.3.1 Outline of the Algorithm

Before we describe the algorithm, we give a high level outline of its main ideas, and where the double exponential dependence on w in the degree of the polynomial comes from. The overall strategy is very similar to that of the algorithm for b -COLORING presented earlier in this chapter. The algorithm is bottom-up dynamic programming along the given branch decomposition of the input graph. Let t be some node in the branch decomposition. To keep the number of table entries bounded by something that is XP in the module-width, we have to find a way to group color classes into a number of *types* that is upper bounded by a function of w alone. The intention is that two color classes of the same type are interchangeable with respect to the underlying coloring being completable to a valid clique coloring of the whole graph. Partial solutions (colorings of the subgraph G_t) can then be described by remembering, for each type, how many color classes of that type there are. If the number of types is $f(w)$ for some function f , this gives an upper bound of $n^{f(w)}$ on the number of table entries at each node of the branch decomposition.

Let us discuss what kind of information goes into the definition of a type. Since the final coloring of G has to avoid monochromatic maximal cliques, we maintain information about cliques in G_t that are or may become monochromatic maximal cliques in some extension of the coloring at hand. A natural attempt would be to consider and describe maximal cliques in G_t by their intersection patterns with the equivalence classes of \sim_t . However, it is not sufficient to consider only maximal cliques in G_t ; given a maximal clique X in G_t , it may happen that in $\overline{V_t}$ there is a vertex v that is adjacent to a strict subset $Y \subset X$ of that clique, forming a maximal

clique with Y – which does not fully contain X – in a supergraph of G_t . Considering the equivalence classes of \sim_t , this implies that the equivalence classes containing Y and the ones containing $X \setminus Y$ are disjoint. We therefore consider cliques X that are *maximal in the subgraph induced by the equivalence classes containing vertices of X* . We call such cliques X *eqc-maximal*, and observe that with a little extra information, we can keep track of the forming and disintegrating of eqc-maximal cliques along the branch decomposition. If an eqc-maximal clique is fully contained in some set of vertices (/color class) C , then we call it *potentially bad for C* . A potentially bad clique is described via its *profile*, which consists of the intersection pattern with the equivalence classes of \sim_t , and some extra information. At each node, there are at most $2^{\mathcal{O}(w)}$ profiles of such cliques.

Equipped with this definition, we can define the notion of a t -type of a color class C , which is simply the subset of profiles at t , such that G_t contains a potentially bad clique with that C -profile. It immediately follows that the number of t -types is $2^{2^{\mathcal{O}(w)}}$. Now, colorings \mathcal{C}_t of G_t are described by their *t -signature*, which records how many color classes of each type \mathcal{C}_t has. There are at most $k^{f(w)}$ many t -signatures, where $f(w) = 2^{2^{\mathcal{O}(w)}}$, and up to some constants in the degree of the polynomial, this essentially upper bounds the runtime of the resulting algorithm by $n \cdot k^{\mathcal{O}(f(w))} = n^{\mathcal{O}(f(w))}$.

At the root node $\mathbf{r} \in V(T)$, there is only one equivalence class, namely $V_{\mathbf{r}} = V(G)$, and if in a coloring, there is a clique that is potentially bad for some color class, then it is indeed a monochromatic maximal clique. Therefore, at the root node, we only have to check whether there is a coloring all of whose color classes have no potentially bad cliques.

10.3.2 Potentially Bad Cliques

We now introduce the main concept used to describe color classes in partial solutions of our algorithms, namely potentially bad cliques. These are cliques that are monochromatic in some subgraph induced by a set of equivalence classes.⁸

Definition 10.23 (Potentially Bad Clique). Let G be a graph with rooted branch decomposition (T, \mathcal{L}) and let $t \in V(T)$. A clique X in G_t is called *eqc-maximal (in G_t)* if it is maximal in $G_t[V(\text{eqc}_t(X))]$. Let $C \subseteq V_t$ and let X be a clique in G_t . Then, X is called *potentially bad for C (in G_t)*, if X is eqc-maximal in G_t and $X \subseteq C$.

Naturally, it is not feasible to keep track of *all* potentially bad cliques. We therefore capture the most vital information about potentially bad cliques in the following notion of a *profile*. For our algorithm, it is only important to know for a color class whether or not it has some potentially bad clique with a given profile,

⁸Recall that for $t \in V(T)$, and a set $S \subseteq V_t$, $\text{eqc}_t(S)$ is the set of equivalence classes of \sim_t with a nonempty intersection with S .

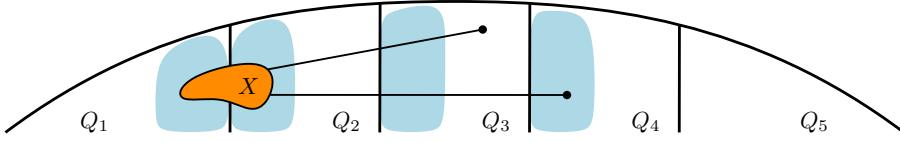


Figure 10.3: Illustration of the C -profile of a clique X that is potentially bad for a color class C , depicted as the shaded areas within the equivalence classes. In this case, we have that $\pi(X \mid C) = (\{Q_1, Q_2\}, \{Q_3, Q_4\})$.

rather than how many, or what its vertices are. This is key to reduce the amount of information we need to store about partial solutions.

There are two components of a profile of a potentially bad clique X ; the first one is the set of equivalence classes \mathcal{Q} containing its vertices, and the second one consists of the equivalence classes $P \notin \mathcal{Q}$ that have a vertex that is complete to X . This is because, at a later stage, P may be merged with an equivalence class containing vertices of X (via the bubbles), in which case X is no longer potentially bad. We illustrate the following definition in Figure 10.3.

Definition 10.24 (Profile). Let G be a graph with rooted branch decomposition (T, \mathcal{L}) and let $t \in V(T)$. Let $C \subseteq V_t$ and let X be a clique in G_t that is potentially bad for C . The C -profile of X is a pair of subsets of V_t/\sim_t , $\pi(X \mid C) := (\mathcal{Q}, \mathcal{P})$, where

$$\mathcal{Q} = \text{eqc}_t(X) \text{ and } \mathcal{P} = \{P \in \overline{\text{eqc}}_t(X) \mid \exists v \in P: X \subseteq N(v)\}.$$

We call the set of all pairs of disjoint subsets of V_t/\sim_t , where the first coordinate is nonempty, the *profiles at t*, formally,

$$\Pi_t := \{(\mathcal{Q}, \mathcal{P}) \mid \mathcal{Q}, \mathcal{P} \subseteq V_t/\sim_t: \mathcal{Q} \neq \emptyset \wedge \mathcal{Q} \cap \mathcal{P} = \emptyset\}.$$

Observation 10.25. Let (T, \mathcal{L}) be a rooted branch decomposition. For each $t \in V(T)$, there are at most $2^{\mathcal{O}(w)}$ profiles at t , where $w = \text{mw}(T, \mathcal{L})$.

Let $t \in V(T) \setminus L(T)$ be an internal node with children r and s and operator (H_t, η_r, η_s) , and let $\pi_r \in \Pi_r$ and $\pi_s \in \Pi_s$ be a pair of profiles. We are now working towards a notion that precisely captures when and how a potentially bad clique in G_r for some $C_r \subseteq V_r$ with C_r -profile π_r can be merged with a potentially bad clique in G_s for some $C_s \subseteq V_s$ with C_s -profile π_s to obtain a potentially bad clique for $C_r \cup C_s$ in G_t . As it turns out, if this is possible, then the profile of the resulting clique only depends on π_r , π_s , and the operator of t . Note that for now, we focus on the case when the cliques in G_r and G_s are both nonempty, and we discuss the case when one of them is empty below.

Before we proceed with this description, we need to introduce some more concepts. We illustrate all of the following concepts in Figure 10.4. For a set of equivalence

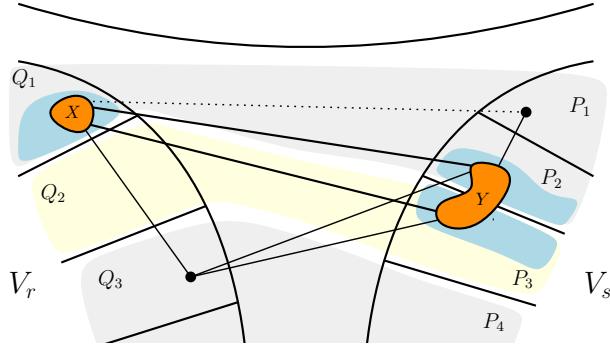


Figure 10.4: Merging a potentially bad clique X in G_r with a potentially bad clique Y in G_s to obtain a potentially bad clique in G_t . The color class at hand is depicted in blue and the gray and yellow areas show the (three) bubbles. Note that the equivalence classes P_1 and Q_2 are bubble buddies of $\text{eqc}_r(X)$ and $\text{eqc}_s(Y)$. Moreover, the types of X and Y are compatible, since $\{Q_1, P_2, P_3\}$ is a maximal biclique in $H_t[\{Q_1, P_1, P_2, P_3\}]$. Finally, note that the equivalence class of \sim_t corresponding to the bubble containing Q_3 will have a vertex that is complete to the potentially bad clique $X \cup Y$.

classes $\mathcal{S} \subseteq V_r/\sim_r \cup V_s/\sim_s$, its *bubble buddies at t*, denoted by $\text{bb}_t(\mathcal{S})$, are the equivalence classes of $V_r/\sim_r \cup V_s/\sim_s$ that are in the same bubble as some equivalence class in \mathcal{S} :

$$\text{bb}_t(\mathcal{S}) := \bigcup_{p \in \{r, s\}} \{Q_p \in V_p/\sim_p \mid \eta_p(Q_p) \in \eta_p(\mathcal{S} \cap V_p/\sim_p)\}$$

We say that $\pi_r = (\mathcal{Q}_r, \mathcal{P}_r)$ and $\pi_s = (\mathcal{Q}_s, \mathcal{P}_s)$ are *compatible*, if $\mathcal{Q}_r \cup \mathcal{Q}_s$ is a maximal biclique in

$$H'_t(\pi_r, \pi_s) := H_t[(\mathcal{Q}_r \cup \mathcal{Q}_s) \cup ((\mathcal{P}_r \cup \mathcal{P}_s) \cap \text{bb}_t(\mathcal{Q}_r \cup \mathcal{Q}_s))]. \quad (10.1)$$

As we show below, the notion of compatibility precisely captures the ‘merging behavior’ of potentially bad cliques. Moreover, for π_r and π_s compatible, we can immediately construct the profile of the resulting potentially bad clique: the *merge profile* of π_r and π_s is the profile $\mu(\pi_r, \pi_s) = (\mathcal{Q}_t, \mathcal{P}_t)$ such that

- $\mathcal{Q}_t = \eta_r(\mathcal{Q}_r) \cup \eta_s(\mathcal{Q}_s)$ and
- $\mathcal{P}_t = \bigcup_{\{o, p\} = \{r, s\}} \{\eta(Q_p) \mid Q_p \in \mathcal{P}_p \setminus \text{bb}_t(\mathcal{Q}_r \cup \mathcal{Q}_s) : \mathcal{Q}_o \subseteq N_{H_t}(Q_p)\}$.

Lemma 10.26. *Let $t \in V(T) \setminus L(T)$ be an internal node with children r and s and operator (H_t, η_r, η_s) . For all $p \in \{r, s\}$, let $C_p \subseteq V_p$, let X_p be a clique in G_r that is potentially bad for C_p , and let $\pi_p := \pi(X_p \mid C_p) = (\mathcal{Q}_p, \mathcal{P}_p)$. If π_r and π_s are compatible, then $X_t := X_r \cup X_s$ is a clique that is potentially bad for $C_t := C_r \cup C_s$, and $\pi(X_t \mid C_t) = \mu(\pi_r, \pi_s)$.*

Proof. We first argue that X_t is a clique. Since X_r and X_s are cliques, we only have to show that for each $v_r \in X_r$ and $v_s \in X_s$, $v_r v_s \in E(G_t)$. In other words, if Q_r is the equivalence class of \sim_r containing v_r , and Q_s is the equivalence class of \sim_s containing v_s , then $Q_r Q_s \in E(H_t)$. Now, $Q_r \in \text{eqc}_r(X_r) = \mathcal{Q}_r$ and $Q_s \in \text{eqc}_s(X_s) = \mathcal{Q}_s$, and since π_r and π_s are compatible, we have that $\mathcal{Q}_r \cup \mathcal{Q}_s$ is a biclique in H_t , therefore $Q_r Q_s \in E(H_t)$.

Next, we show that X_t is potentially bad for C_t . Since X_r and X_s are potentially bad for C_r and C_s , respectively, we have that $X_r \subseteq C_r$ and $X_s \subseteq C_t$, and therefore $X_t = X_r \cup X_s \subseteq C_r \cup C_s = C_t$. It remains to show that X_t is eqc-maximal. Suppose not, and let $y \in V(\text{eqc}_t(X_t))$ be a vertex that is complete to X_t . First, we know that $y \notin V(\text{eqc}_r(X_r) \cup \text{eqc}_s(X_s))$, for if $y \in V(\text{eqc}_p(X_p))$ for some $p \in \{r, s\}$, then X_p is not eqc-maximal, contradicting X_p being potentially bad for C_p . On the other hand, we have that $\text{eqc}_t(X_t) = \text{bb}_t(\text{eqc}_r(X_r) \cup \text{eqc}_s(X_s)) = \text{bb}_t(\mathcal{Q}_r \cup \mathcal{Q}_s)$. We may assume that for some $p \in \{r, s\}$, the vertex y is contained in some $Q_p \in \text{bb}_t(\mathcal{Q}_r \cup \mathcal{Q}_s) \setminus (\mathcal{Q}_r \cup \mathcal{Q}_s)$. Assume up to renaming that $p = r$. Since y is complete to X_t , we have that y is complete to X_r , and therefore $Q_r \in \mathcal{P}_r$. In other words, Q_r is contained in the graph $H'_t(\pi_r, \pi_s)$ as described in Equation (10.1). Moreover, since y is complete to X_s , we have that Q_r is complete to $\text{eqc}_s(X_s) = \mathcal{Q}_s$. This implies that $\{Q_r\} \cup \mathcal{Q}_r \cup \mathcal{Q}_s$ is a biclique in $H'_t(\pi_r, \pi_s)$, contradicting π_r and π_s being compatible.

To conclude the proof, we need to show that $\pi(X_t \mid C_t) = \mu(\pi_r, \pi_s)$. Let $\mu(\pi_r, \pi_s) = (\mathcal{Q}_t, \mathcal{P}_t)$. We first show that $\text{eqc}_t(X_t) = \mathcal{Q}_t$. To see that $\mathcal{Q}_t = \eta_r(\mathcal{Q}_r) \cup \eta_s(\mathcal{Q}_s) \subseteq \text{eqc}_t(X_t)$, we observe that for all $Q_p \in \mathcal{Q}_p$, there is an $x \in X_p \cap Q_p$. This means that $x \in \eta_p(Q_p)$, therefore $X_t \cap \eta_p(Q_p) \neq \emptyset$ and $\eta_p(Q_p) \in \text{eqc}_t(X_t)$. The other inclusion can be argued similarly.

Now suppose that $Q_t \in \mathcal{P}_t$. Then, $Q_t = \eta_p(Q_p)$ for some $Q_p \in \mathcal{P}_p \setminus \text{bb}_t(\mathcal{Q}_r \cup \mathcal{Q}_s)$ with $\mathcal{Q}_o \subseteq N_{H_t}(Q_p)$. In other words, there is a vertex $v \in Q_p$ that is complete to X_t , and $\eta_p(Q_p) \notin \text{eqc}_t(X_t)$. According to the definition of a profile, $Q_t = \eta_p(Q_p)$ is contained in the second coordinate of π_t . The other inclusion can be shown similarly. \square

Now we show the other direction, i.e. that if we have a potentially bad clique for some $C_t \subseteq V_t$ in G_t , then its restrictions to V_r and V_s necessarily also form potentially bad cliques for the restriction of C_t to V_r and V_s in G_r and G_s , respectively. Furthermore, in that case, the profiles of the resulting cliques are compatible.

Lemma 10.27. *Let $t \in V(T) \setminus L(T)$ be an internal node with children r and s and operator (H_t, η_r, η_s) . Let $C_t \subseteq V_t$, and let X_t be a clique in G_t that is potentially bad for C_t . For all $p \in \{r, s\}$, let $X_p := X_t \cap V_p$ and $C_p := C_t \cap V_p$. Suppose that for all $p \in \{r, s\}$, $X_p \neq \emptyset$. Then, for all $p \in \{r, s\}$, X_p is a potentially bad clique for C_p , and $\pi_r := \pi(X_r \mid C_r)$ and $\pi_s := \pi(X_s \mid C_s)$ are compatible.*

Proof. Since X_t is a potentially bad clique for C_t , we have that $X_t \subseteq C_t$, and so for $p \in \{r, s\}$, $X_p \subseteq C_p$. It remains to show that X_p is eqc-maximal for all $p \in \{r, s\}$. Up to renaming, it suffices to show that X_r is eqc-maximal. Suppose not and let $y \in \text{eqc}_r(X_r)$ be a vertex that is complete to X_r . Since X_t is a clique in G_t , we have

that $\text{eqc}_r(X_r) \cup \text{eqc}_s(X_s)$ is a biclique in H_t . Therefore, y is also complete to X_s and therefore to X_t . Clearly, $y \in \text{eqc}_t(X_t)$, and we have a contradiction with X_t being eqc-maximal.

What remains to be shown is that $\pi_r = (\mathcal{Q}_r, \mathcal{P}_r)$ and $\pi_s = (\mathcal{Q}_s, \mathcal{P}_s)$ are compatible. We have already argued that $\mathcal{Q}_r \cup \mathcal{Q}_s = \text{eqc}_r(X_r) \cup \text{eqc}_s(X_s)$ is a biclique in H_t ; we have to show that $\mathcal{Q}_r \cup \mathcal{Q}_s$ is a maximal biclique in $H'_t := H'_t(\pi_r, \pi_s)$ as defined in Equation (10.1). Clearly, $\mathcal{Q}_r \cup \mathcal{Q}_s \subseteq V(H'_t)$, so suppose that $\mathcal{Q}_r \cup \mathcal{Q}_s$ is not a maximal biclique in H'_t . This means that for some $p \in \{r, s\}$, there is some $Q_p \in \mathcal{P}_p \cap \text{bb}_t(\mathcal{Q}_r \cup \mathcal{Q}_s)$ such that $\{Q_p\} \cup \mathcal{Q}_r \cup \mathcal{Q}_s$ is a biclique in H'_t . In that case, there is a vertex $y \in Q_p$ that is complete to X_t (since $Q_p \in \mathcal{P}_p$ and $\{Q_p\} \cup \mathcal{Q}_r \cup \mathcal{Q}_s$ is a biclique), and $y \in V(\text{eqc}_t(X_t))$ (since $Q_p \in \text{bb}_t(\mathcal{Q}_r \cup \mathcal{Q}_s)$); we obtained a contradiction with X_t being eqc-maximal. \square

As mentioned above, we treat the case when a clique X_p in one of the children $p \in \{r, s\}$ remains potentially bad in G_t separately. This is because in that case, the notion of a maximal biclique in H'_t as defined in Equation (10.1) does not hold up very naturally. We formulate the analogous requirements for this case here, and we skip some of the details.

Let $t \in V(T) \setminus L(T)$ be an internal node with children r and s and operator (H_t, η_r, η_s) . Let $\pi_r \in \Pi_r$. We say that π_r is *liftable* if

- there is no $Q_s \in \text{bb}_t(\mathcal{Q}_r)$ that is complete to \mathcal{Q}_r in H_t , and
- $\text{bb}_t(\mathcal{Q}_r) \cap \mathcal{P}_r = \emptyset$.

The *lift profile* of π_r , denoted by $\lambda(\pi_r)$, is constructed as the merge profile of π_r with the empty set; i.e. we take $(\mathcal{Q}_s, \mathcal{P}_s) = (\emptyset, V_s / \sim_s)$ and apply the definition given above.

Lemma 10.28. *Let $t \in V(T) \setminus L(T)$ be an internal node with children r and s . Let $C_r \subseteq V_r$, $C_s \subseteq V_s$, let X_r be a clique in G_r , and let $\pi_r := \pi(X_r \mid C_r)$. Then, X_r is a potentially bad clique for $C_r \cup C_s$ in G_t if and only if X_r is a potentially bad clique for C_r in G_r and π_r is liftable, in which case $\pi_t(X_r \mid C_r \cup C_s) = \lambda(\pi_r)$.*

Proof. The proof can be done with very similar arguments to those given above and is therefore omitted. One only needs to observe that the notion of ‘liftable’ modulates the notion of a profile being compatible with the profile of an empty set. \square

10.3.3 The Type of a Color Class

We now describe the t -type of a color class C , which is the subset of profiles at t such that there is a clique in G_t that is potentially bad for C , with that C -profile. For our algorithm, two color classes with the same type will be interchangeable, therefore we only have to remember the number of color classes of each type.

Definition 10.29 (*t*-Type). Let G be a graph with rooted branch decomposition (T, \mathcal{L}) , and let $t \in V(T)$. For a set $C \subseteq V_t$, the *t-type* of C , denoted by $\gamma_t(C)$ is

$$\gamma_t(C) := \{\pi_t \in \Pi_t \mid \exists \text{ clique } X \text{ in } G_t \text{ which is potentially bad for } C \text{ and} \\ \pi(X \mid C) = \pi_t\}.$$

We call the set $\Gamma_t = 2^{\Pi_t}$ of all subsets of profiles at t the *t-types*.

Since for each $t \in V(T)$, $|\Pi_t| \leq 2^{\mathcal{O}(w)}$ by Observation 10.25, the number of *t*-types can be upper bounded as follows.

Observation 10.30. Let (T, \mathcal{L}) be a rooted branch decomposition, and let $t \in V(T)$. There are at most $2^{2^{\mathcal{O}(w)}}$ many *t*-types, where $w := \text{mw}(T, \mathcal{L})$.

In our algorithm we want to be able to determine the *t*-type of the union of a color class in G_r and a color class in G_s . This is done via the following notion of a merge type, which is based on the notion of merge and lift profiles given in the previous section.

Definition 10.31 (Merge Type). Let G be a graph with rooted branch decomposition (T, \mathcal{L}) , let $t \in V(T) \setminus L(T)$ with children r and s . For a pair of an r -type $\gamma_r \in \Gamma_r$ and an s -type $\gamma_s \in \Gamma_s$, the *merge type* of γ_r and γ_s , denoted by $\mu(\gamma_r, \gamma_s)$, is the *t*-type obtained as follows.

$$\mu(\gamma_r, \gamma_s) := \{\mu(\pi_r, \pi_s) \mid \pi_r \in \gamma_r, \pi_s \in \gamma_s, \text{where } \pi_r \text{ and } \pi_s \text{ are compatible}\} \\ \bigcup_{p \in \{r, s\}} \{\lambda(\pi_p) \mid \pi_p \in \gamma_p, \text{where } \pi_p \text{ is liftable}\}$$

Lemma 10.32. Let G be a graph with rooted branch decomposition (T, \mathcal{L}) , let $t \in V(T) \setminus L(T)$ with children r and s . Let $C_r \subseteq V_r$ and $C_s \subseteq V_s$. Then, $\gamma_t(C_r \cup C_s) = \mu(\gamma_r(C_r), \gamma_s(C_s))$.

Proof. Let $C_t := C_r \cup C_s$. For one inclusion, let $\pi_t \in \gamma_t(C_t)$. Then, there is a clique X_t in G_t that is potentially bad for C_t whose C_t -profile is π_t . If for all $p \in \{r, s\}$, $X_p := X_t \cap V_p \neq \emptyset$, then by Lemma 10.27, we know that for all $p \in \{r, s\}$, X_p is a potentially bad clique for $C_p := C_t \cap V_p$, therefore $\pi_p := \pi(X_p \mid C_p) \in \gamma_p(C_p)$. Moreover, the lemma asserts that π_r and π_s are compatible, so by construction, we can conclude that $\pi_t = \mu(\pi_r, \pi_s) \in \mu(\gamma_r(C_r), \gamma_s(C_s))$. On the other hand, if for some $p \in \{r, s\}$, $X_t \subseteq V_p$, then by Lemma 10.28, X_t is a potentially bad clique for C_p , so $\pi_p := \pi_p(X_t \mid C_p) \in \gamma_p(C_p)$. The lemma also asserts that π_p is liftable and that $\lambda(\pi_r) = \pi_t$, in which case we also have that $\pi_t \in \mu(\gamma_r(C_r), \gamma_s(C_s))$. We have argued that $\gamma_t(C_t) \subseteq \mu(\gamma_r(C_r), \gamma_s(C_s))$.

For the other inclusion, suppose that $\pi_t \in \mu(\gamma_r(C_r), \gamma_s(C_s))$. Then, either there is a pair of profiles $\pi_r \in \gamma_r(C_r)$, $\pi_s \in \gamma_s(C_s)$ such that π_r and π_s are compatible and $\pi_t = \mu(\pi_r, \pi_s)$ or for some $p \in \{r, s\}$, there is a profile $\pi_p \in \gamma_p(C_p)$ that is liftable and $\pi_t = \lambda(\pi_p)$. In the former case, we can use Lemma 10.26 to conclude that $\pi_t \in \gamma_t(C_t)$, and in the latter case, we have that $\pi_t \in \gamma_t(C_t)$ by Lemma 10.28. This shows that $\mu(\gamma_r(C_r), \gamma_s(C_s)) \subseteq \gamma_t(C_t)$ which concludes the proof. \square

10.3.4 The Algorithm

We are now ready to describe the algorithm. As alluded to above, partial solutions at a node t , i.e. colorings of G_t , are described via the notion of a *t -signature* which records the number of color classes of each type in a coloring. If two colorings have the same t -signature, then they are interchangeable as far as our algorithm is concerned. We show that this information suffices to solve the problem in a bottom-up dynamic programming fashion.

Definition 10.33 (t -Signature). Let k be a positive integer. Let G be a graph with rooted branch decomposition (T, \mathcal{L}) , let $t \in V(T)$, and let $\mathcal{C} = (C_1, \dots, C_k)$ be a k -coloring of G_t . Then, $\text{sig}_{\mathcal{C}} : \Gamma_t \rightarrow \{0, 1, \dots, k\}$ where

$$\forall \gamma_t \in \Gamma_t : \text{sig}_{\mathcal{C}}(\gamma_t) := |\{i \in \{1, \dots, k\} \mid \gamma(C_i) = \gamma_t\}|,$$

is called the *t -signature* of \mathcal{C} . The set of *t -signatures* is defined as:

$$\text{SIG}_t := \left\{ \text{sig}_t : \Gamma_t \rightarrow \{0, 1, \dots, k\} \mid \sum_{\gamma_t \in \Gamma_t} \text{sig}_t(\gamma_t) = k \right\}$$

The following bound on the number of t -signatures immediately follows from Observation 10.30, stating that the number of t -types is upper bounded by $2^{2^{\mathcal{O}(w)}}$.

Observation 10.34. Let (T, \mathcal{L}) be a rooted branch decomposition of an n -vertex graph, and let $t \in V(T)$. There are at most $k^{2^{\mathcal{O}(w)}} \leq n^{2^{\mathcal{O}(w)}}$ many t -signatures, where $w := \text{mw}(T, \mathcal{L})$ and k is the number of colors.

Definition of the table entries. For each $t \in V(T)$ and each $\text{sig}_t \in \text{SIG}_t$, we let $\text{tab}[t, \text{sig}_t] = 1$ if and only if there is a k -coloring \mathcal{C} of G_t such that $\text{sig}_{\mathcal{C}} = \text{sig}_t$. \square

We now show that the information stored at the table entries suffices to determine whether or not our input is a YES-instance; that is, after filling all the table entries, we can read off the solution to the problem at the root node.

Lemma 10.35. *Let G be a graph with rooted branch decomposition (T, \mathcal{L}) , and let \mathfrak{r} be the root of T . G has a clique coloring with k colors if and only if $\text{tab}[\mathfrak{r}, \text{sig}^*] = 1$, where sig^* is the \mathfrak{r} -signature for which $\text{sig}^*(\emptyset) = k$.*

Proof. The lemma immediately follows from two facts. First, since $\text{sig}^*(\emptyset) = k$, we have that $\text{sig}^*(\gamma_{\mathfrak{r}}) = 0$ for any other \mathfrak{r} -type $\gamma_{\mathfrak{r}} \neq \emptyset$. Second, that for each set $C \subseteq V_{\mathfrak{r}} = V(G)$, the set of potentially bad cliques for C is precisely the set of maximal cliques that are fully contained in C , i.e. it is the set of monochromatic maximal cliques in the corresponding coloring that are contained in C . \square

We first describe how to compute the table entries at the leaves, by brute-force.

Leaves of T . Let $t \in L(T)$ be a leaf node in T and let $v \in V(G)$ be the vertex such that $\mathcal{L}(v) = t$. We show how to compute the table entries $\text{tab}[t, \cdot]$. Note that $G_t = (\{v\}, \emptyset)$, and that $\{v\}$ is the only equivalence class of \sim_t . To describe the types of color classes of G_t , observe that the only eqc-maximal clique in G_t is $\{v\} =: X_v$, which is potentially bad for $C_v := \{v\} = X_v$. In that case, we have that $\pi_v := \pi(X_v \mid C_v) = (\{v\}, \emptyset)$, and the type of color class C_v is $\{\pi_v\}$. The type of the remaining $k - 1$ color classes is \emptyset , since they are all empty. Therefore, for each t -signature sig_t , we set $\text{tab}[t, \text{sig}_t] := 1$ if and only if $\text{sig}_t(\{\pi_v\}) = 1$ and $\text{sig}_t(\emptyset) = k - 1$. \triangleleft

Next, we move on to the computation of the table entries at internal nodes of the branch decomposition. To describe this part of the algorithm, we adapt the notion of a *merge skeleton* from Section 10.1. In the algorithm we present here, we do not require⁹ the notion of compatibility of types, therefore the bipartite graph in a merge skeleton for CLIQUE COLORING is complete.

Definition 10.36 (Merge skeleton). Let G be a graph and (T, \mathcal{L}) one of its rooted branch decompositions. Let $t \in V(T) \setminus L(T)$ with children r and s . The *merge skeleton* of r and s is an edge-labeled complete bipartite graph $(\mathfrak{J}, \mathfrak{m})$ where

- $V(\mathfrak{J}) = \Gamma_r \cup \Gamma_s$, and
- for all $\gamma_r \in \Gamma_r, \gamma_s \in \Gamma_s$, $\mathfrak{m}(\gamma_r \gamma_s) = \mu(\gamma_r, \gamma_s)$.

```

1 foreach  $\text{sig}_t \in \text{SIG}_t$  do set  $\text{tab}[t, \text{sig}_t] \leftarrow 0$ ;
2 Let  $(\mathfrak{J}, \mathfrak{m})$  be the merge skeleton of  $t$ ;
3 foreach  $\text{sig}_r \in \text{SIG}_r, \text{sig}_s \in \text{SIG}_s$  such that  $\text{tab}[r, \text{sig}_r] = 1$  and  $\text{tab}[s, \text{sig}_s] = 1$ 
    do
4     foreach  $\mathfrak{n}: E(\mathfrak{J}) \rightarrow \{0, 1, \dots, k\}$  such that
            (i)  $\sum_{e \in E(\mathfrak{J})} \mathfrak{n}(e) = k$ , and
            (ii) for all  $\{o, p\} = \{r, s\}$  and all  $\gamma_p \in \Gamma_p$ ,  $\sum_{\gamma_p \gamma_o \in E(\mathfrak{J})} \mathfrak{n}(\gamma_p \gamma_o) = \text{sig}_p(\gamma_p)$ 
5     do
6         Let  $\text{sig}_t: \Gamma_t \rightarrow \{0, 1, \dots, k\}$  be such that for all  $\gamma_t \in \Gamma_t$ ,
             $\text{sig}_t(\gamma_t) = \sum_{e \in E(\mathfrak{J}), \mathfrak{m}(e)=\gamma_t} \mathfrak{n}(e)$ ;
7         update  $\text{tab}[t, \text{sig}_t] \leftarrow 1$ ;

```

Algorithm 6: Algorithm to set the table entries at an internal node $t \in V(T) \setminus L(T)$ with children r and s , assuming the table entries at r and s have been computed.

⁹For instance, in CLIQUE COLORING, we do not require the color classes to be independent sets, which is one reason why we do not need a notion of compatibility.

Internal nodes of T . Let $t \in V(T) \setminus L(T)$ be an internal node with children r and s . We discuss how to compute the table entries at t , assuming the table entries at r and s have been computed. Each coloring of G_t can be obtained from a coloring of G_r and a coloring of G_s , by merging pairs of color classes. Therefore, for each pair $\text{sig}_r \in \text{SIG}_r$, $\text{sig}_s \in \text{SIG}_s$ such that $\text{tab}[r, \text{sig}_r] = 1$ and $\text{tab}[s, \text{sig}_s] = 1$, we do the following. We enumerate all labelings of the edge set of the merge skeleton with numbers from $\{0, 1, \dots, k\}$, with the following interpretation. If an edge $\gamma_r \gamma_s$ has label j , then it means that j color classes of r -type γ_r will be merged with j color classes of s -type γ_s ; this gives j color classes of t -type $\mu(\gamma_r, \gamma_s) = m(\gamma_r \gamma_s)$. Each such labeling that respects the number of color classes available of each type will produce a coloring of G_t with some signature sig_t , which can then be read off the edge labeling. For all such sig_t , we set $\text{tab}[t, \text{sig}_t] = 1$. We give the formal details in Algorithm 6. \triangleleft

We now prove correctness of the algorithm.

Lemma 10.37. *Let G be a graph and (T, \mathcal{L}) one of its rooted branch decompositions, and let $t \in V(T)$. The above algorithm computes the table entries $\text{tab}[t, \cdot]$ correctly, i.e. for each $\text{sig}_t \in \text{SIG}_t$, it sets $\text{tab}[t, \text{sig}_t] = 1$ if and only if G_t has a k -coloring \mathcal{C} with $\text{sig}_{\mathcal{C}} = \text{sig}_t$.*

Proof. The proof is by induction on the height of t . In the base case, when t is a leaf, it is straightforward to verify correctness.

Now suppose that $t \in V(T) \setminus L(T)$ is an internal node with children r and s , and let (\mathfrak{J}, m) be the merge skeleton at t . Suppose for some t -signature $\text{sig}_t \in \text{SIG}_t$, the algorithm set $\text{tab}[t, \text{sig}_t] = 1$. Then, there is some r -signature sig_r and some s -signature sig_s such that $\text{tab}[r, \text{sig}_r] = 1$, $\text{tab}[s, \text{sig}_s] = 1$, and there is a map $n: E(\mathfrak{J}) \rightarrow \{0, 1, \dots, k\}$ satisfying the conditions of lines 4 and 6 in Algorithm 6. By induction, there is a k -coloring \mathcal{C}_r of G_r whose r -signature is sig_r , and a k -coloring of \mathcal{C}_s of G_s whose s -signature is sig_s . We construct the desired coloring \mathcal{C}_t of G_t whose t -signature is sig_t as follows: For each pair of an r -type γ_r and an s -type γ_s , we take $n(\gamma_r \gamma_s)$ pairs of a color class C_r of r -type sig_r and a color class C_s of s -type sig_s , and for each such pair, we add $C_r \cup C_s$ as a color class to \mathcal{C}_t . By Lemma 10.32, the t -type of $C_r \cup C_s$ is $\mu(\gamma_r, \gamma_s) = m(\gamma_r \gamma_s)$. The condition in line 4 ensures that each color class of \mathcal{C}_r and each color class of \mathcal{C}_s is used precisely once to create a color class of \mathcal{C}_t , and the condition in line 6 ensures that the t -signature of \mathcal{C}_t is indeed sig_t .

For the other direction, suppose that there is a k -coloring \mathcal{C}_t of G_t with t -signature sig_t . We construct a pair of a coloring \mathcal{C}_r of G_r and a coloring of \mathcal{C}_s of G_s , together with their signatures sig_r and sig_s , respectively, and a map $n: E(\mathfrak{J}) \rightarrow \{0, 1, \dots, k\}$. Initially, for all $p \in \{r, s\}$, we let $\mathcal{C}_p = \emptyset$, and for all $\gamma_p \in \Gamma_p$, $\text{sig}_p(\gamma_p) := 0$. Moreover, we let $n(e) := 0$ for all $e \in E(\mathfrak{J})$.

For each color class $C_t \in \mathcal{C}_t$, we add $C_r := C_t \cap V_r$ to \mathcal{C}_r and $C_s := C_t \cap V_s$ to \mathcal{C}_s . Let γ_t be the t -type of C_t . By Lemma 10.32, C_r has some r -type γ_r and C_s has some s -type γ_s such that γ_t is the merge type $\mu(\gamma_r, \gamma_s)$ of γ_r and γ_s . We increase the

values of $\text{sig}_r(\gamma_r)$ and $\text{sig}_s(\gamma_s)$ by 1, since we added one more color class of r -type γ_r to \mathcal{C}_r , and one more color class of s -type γ_s to \mathcal{C}_s . Additionally, we add 1 to the value of $\mathbf{n}(\gamma_r \gamma_s)$, since C_t is a color class of t -type $\mu(\gamma_r, \gamma_s) = \mathbf{m}(\gamma_r \gamma_s)$ obtained from merging C_r (a color class of r -type γ_r) with C_s (a color class of s -type γ_s).

After doing this for all color classes of \mathcal{C}_t , we have that \mathcal{C}_r is a k -coloring with r -signature sig_r , and that \mathcal{C}_s is a k -coloring with s -signature sig_s . By induction, $\text{tab}[r, \text{sig}_r] = 1$ and $\text{tab}[s, \text{sig}_s] = 1$. It remains to argue that \mathbf{n} satisfies the conditions expressed in lines 4 and 6 in Algorithm 6. The first item of line 4 is clearly satisfied, since we increased $|\mathcal{C}_t| = k$ values of \mathbf{n} by 1 in the above process. The second item holds since we increased the value of some $\text{sig}_p(\gamma_p)$ by 1 if and only if we increased the value of an edge e incident with γ_p in \mathfrak{J} by 1. To see that for each γ_t , $\text{sig}_t(\gamma_t) = \sum_{e \in E(\mathfrak{J}), \mathbf{m}(e)=\gamma_t} \mathbf{n}(e)$, observe that we identified for each color class of type γ_t , the occurrence of γ_t as a merge type of a pair of an r -type and an s -type, and therefore a label of some edge $e \in E(\mathfrak{J})$, and increased $\mathbf{n}(e)$ by 1 in such a case. We can conclude that sig_t can be obtained as shown in line 6 of Algorithm 6, and so the algorithm set $\text{tab}[t, \text{sig}_t] = 1$. \square

To wrap up, it remains to argue the runtime of the algorithm. Suppose we are given a graph G with rooted branch decomposition (T, \mathcal{L}) and let $w := \text{mw}(T, \mathcal{L})$. By Observation 10.34, there are at most $k^{2^{\mathcal{O}(w)}}$ table entries at each node of T . The entries of leaf nodes can clearly be computed in time $k^{\mathcal{O}(1)}$. Now let $t \in V(T) \setminus L(T)$ be an internal node with children r and s . To compute all table entries at t , we execute Algorithm 6. In the worst case, it loops over each pair of an r -signature and an s -signature, and given such a pair, it enumerates all labelings of the edges of the merge skeleton \mathfrak{J} with numbers from $\{0, 1, \dots, k\}$ (such that all entries sum up to k). We have that $|E(\mathfrak{J})| = |\Gamma_r| \cdot |\Gamma_s| = \left(2^{2^{\mathcal{O}(w)}}\right)^2 = 2^{2^{\mathcal{O}(w)}}$ (see Observation 10.30), therefore the number of labelings to consider is upper bounded by $k^{2^{\mathcal{O}(w)}}$. The runtime of Algorithm 6 can therefore be upper bounded by

$$\left(k^{2^{\mathcal{O}(w)}}\right)^2 \cdot k^{2^{\mathcal{O}(w)}} = k^{2^{\mathcal{O}(w)}},$$

and since $|V(T)| = \mathcal{O}(n)$, the entire algorithm takes time $k^{2^{\mathcal{O}(w)}} \cdot n$. Correctness is proved in Lemma 10.37, and Lemma 10.35 asserts that the solution to the problem can be read off the table entries at the root, once computed. Using standard memoization techniques, we can modify the above algorithm so that it returns a coloring if one exists. We therefore have the following theorem.

Theorem 10.38. *There is an algorithm that given an n -vertex graph G together with one of its rooted branch decompositions (T, \mathcal{L}) and a positive integer k , decides whether G has a clique coloring with k colors in time $k^{2^{\mathcal{O}(w)}} \cdot n$, where $w := \text{mw}(T, \mathcal{L})$. If such a coloring exists, the algorithm can construct it.*

10.4 Conclusion and Open Problems

We conclude this chapter by listing some open problems regarding the complexity of coloring problems parameterized by clique-width. Throughout, we denote the number of vertices of the input graph of the respective problem by n , its clique-width by cw , and the number of colors in the input by k .

In the light of the $\text{W}[1]$ -hardness result and the lower bound for b -COLORING of Proposition 10.1, ruling out $n^{2^{o(\text{cw})}}$ -time algorithms under ETH, it would certainly be a surprise if FALL COLORING admitted an $n^{2^{o(\text{cw})}}$ -time algorithm. Nevertheless, the proof of Proposition 10.1 neither implies $\text{W}[1]$ -hardness for FALL COLORING, nor a lower bound under ETH, therefore we pose the following open problems.

Open Problem 10.1. Is FALL COLORING parameterized by clique-width $\text{W}[1]$ -hard?

Open Problem 10.2. Would an $n^{2^{o(\text{cw})}}$ -time algorithm for FALL COLORING refute ETH?

The situation for CLIQUE COLORING is more intriguing. We provided an algorithm that runs in time $k^{f(\text{cw})} \cdot n$ in Theorem 10.38. CLIQUE COLORING has not been studied through the lens of structural parameterizations before, and there is not even clear evidence that the problem is $\text{W}[1]$ -hard parameterized by clique-width.

Open Problem 10.3. Is CLIQUE COLORING parameterized by clique-width $\text{W}[1]$ -hard?

Hardness proofs for GRAPH COLORING parameterized by clique-width typically rely on the fact that cliques require many colors while keeping the clique-width small; since cliques can be clique colored with two colors, these tricks are of no use in the setting of CLIQUE COLORING.

In the other direction, the most promising route towards an FPT-algorithm for CLIQUE COLORING parameterized by clique-width is a proof that all graphs of clique-width cw can be clique colored with at most $g(\text{cw})$ many colors, for some computable function g . We therefore pose the following problem:

Open Problem 10.4. Is there a computable function $g: \mathbb{N} \rightarrow \mathbb{N}$ such that each graph of clique-width cw can be clique colored with at most $g(\text{cw})$ many colors?

A positive answer to this question would severely reduce the significance of the algorithm from Theorem 10.38. On the other extreme, for all we know, the triple exponential dependence on cw may be the optimal for clique-width based algorithms for CLIQUE COLORING, both when the number of colors is unbounded, and when it is bounded.

Open Problem 10.5. Is there an $n^{2^{2^{o(\text{cw})}}}$ -time algorithm for CLIQUE COLORING or would such an algorithm refute ETH?

Open Problem 10.6. Is there a $2^{2^{o(\text{cw})}}$ -time algorithm for 2-CLIQUE COLORING or would such an algorithm refute ETH?

Next, it would be interesting to see how b -COLORING and FALL COLORING behave on graph classes of bounded mim-width. GRAPH COLORING is known to be NP-complete on circular arc graphs [136], but the standard reductions from GRAPH COLORING to b -COLORING do not preserve the property of a graph being circular arc. So far, all graph classes where b -COLORING is NP-complete are either known to have unbounded mim-width (bipartite [54, 196, 281], co-bipartite [47, 55], chordal [155, 176, 220]) or it is open whether they have unbounded mim-width (line graphs [66]).

Open Problem 10.7. Is b -COLORING NP-complete on circular arc graphs, or, more generally, on any graph class of constant mim-width?

GRAPH COLORING parameterized by the mim-width of a given branch decomposition *plus* the number of colors is XP, since it is an LCVP problem. On the other hand, b -COLORING is not an LCVP problem, so we do not know whether b -COLORING is XP in this parameterization.

Open Problem 10.8. Is b -COLORING parameterized by the mim-width of a given branch decomposition of the input graph plus the number of colors XP?

In 2018, Lampis [202] investigated the fine-grained complexity of GRAPH COLORING with *fixed* number of colors $k \geq 3$, parameterized by the clique-width cw of the input graph. He provided an $\mathcal{O}^*((2^k - 2)^{\text{cw}})$ -time algorithm, and proved that the base of $2^k - 2$ in the exponential of the runtime is tight under SETH: An algorithm running in time $\mathcal{O}^*((2^k - 2 - \epsilon)^{\text{cw}})$, for any $\epsilon > 0$, would falsify SETH.

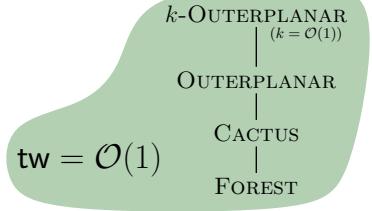
It would be interesting to obtain similar results for b -COLORING. We believe that the base of the exponent of $2^k - 2$ is probably not sufficient for the runtime of an algorithm to solve b -COLORING parameterized by clique-width, for the following reason. Roughly speaking, in the DP-algorithm of Lampis, colorings of subgraphs G_t are described by the subsets of colors appearing on the vertices of each equivalence class of \sim_t . The ‘ -2 ’ is due to the observation that neither the empty set nor the set of all colors need to be considered as parts of table indices. It is reasonable to believe that an algorithm for b -COLORING has to store at least that information as well; and adding any useful information about (partial) b -vertices seems to prohibitively increase the number of table entries at each node.

Open Problem 10.9. For which function $f: \mathbb{N} \rightarrow \mathbb{N}$ does it hold that for all fixed $k \geq 3$, b -COLORING on graphs of clique-width cw can be solved in time $\mathcal{O}^*(f(k)^{\text{cw}})$ but an algorithm running in time $\mathcal{O}^*((f(k) - \epsilon)^{\text{cw}})$, for any $\epsilon > 0$, would refute SETH? What about FALL COLORING, or CLIQUE COLORING?

Lastly, Jaffke et al. [170] also showed that b -COLORING parameterized by the vertex cover number of the input graph is FPT. In terms of expressive power, the parameter treewidth sits between the vertex cover number and the clique-width, therefore it would be interesting to assess the parameterized complexity of b -COLORING

parameterized by treewidth. Our guess is that the problem should be W[1]-hard. In case the problem is indeed W[1]-hard parameterized by treewidth, it would be interesting to give the fastest possible algorithm under ETH. (Note that our clique-width based algorithm only gives an $n^{f(\text{tw})}$ -time algorithm, where $f(\text{tw}) = 2^{2^{\mathcal{O}(\text{tw})}}$, which should not be too difficult to improve upon.)

Open Problem 10.10. Is b -COLORING parameterized by the treewidth of the input graph W[1]-hard? What is the fastest algorithm we can obtain under ETH in this parameterization?



Fine-Grained Complexity of Clique Coloring on Treewidth

It is a standard textbook exercise to give an algorithm for the q -COLORING problem, asking for a proper coloring of the input graph with $q \geq 3$ colors, that runs in time $\mathcal{O}^*(q^{\text{tw}})$, where tw denotes the width of a given tree decomposition of the input graph. Lokshtanov et al. [208] showed that it is unlikely that this runtime can be improved much further. Assuming the Strong Exponential Time Hypothesis (SETH, see Section 2.3), they ruled out the existence of any $\epsilon > 0$ such that there is an $\mathcal{O}^*((q - \epsilon)^{\text{tw}})$ time algorithm for q -COLORING in this parameterization.

In this chapter, we obtain analogous results for the q -CLIQUE COLORING problem in the same parameterization. Recall that a clique coloring with k colors of a graph is an assignment of one of k colors to each of its vertices such that no maximal clique is monochromatic. We show that for any fixed $q \geq 2$, q -CLIQUE COLORING (asking for a clique coloring with q colors) can be solved in time $\mathcal{O}^*(q^{\text{tw}})$, where tw denotes the width of a given tree decomposition of the input graph. We also prove that under SETH, for any $q \geq 2$, there is no $\epsilon > 0$ such that q -CLIQUE COLORING can be solved in time $\mathcal{O}^*((q - \epsilon)^{\text{tw}})$. In fact, we rule out $\mathcal{O}^*((q - \epsilon)^t)$ -time algorithms for a much smaller class of graphs than those of treewidth t , namely: graphs that have both *pathwidth* and *feedback vertex set number* simultaneously bounded by t .

For the upper bound, we give a standard dynamic programming algorithm which uses fast subset convolution [25] to overcome a small runtime barrier. The lower bound is obtained in two parts. For $q \geq 3$, we first use a reduction for q -LIST COLORING parameterized by treewidth that produces triangle-free graphs [167], and then reduce to q -COLORING by adding a gadget based on Mycielski graphs that keeps the graphs triangle-free. Since in a triangle-free graph, the proper colorings are precisely the clique colorings (each maximal clique is an edge), this gives the desired hardness result for at least three colors. Clearly, this approach fails for proving hardness when $q = 2$, since in this case, q -COLORING is polynomial-time solvable.

We therefore give a separate reduction for this case, starting from NAE-SAT.

Notation for Colorings. In this chapter, denote a coloring of a graph with k colors as a map $\phi: V(G) \rightarrow \{1, \dots, k\}$. Such a coloring ϕ is called *proper* if for each pair $u, v \in V(G)$ with $uv \in E(G)$, $\phi(u) \neq \phi(v)$. The restriction of ϕ to a vertex set $S \subseteq V(G)$ is the map $\phi|_S: S \rightarrow \{1, \dots, k\}$ with $\phi|_S(v) = \phi(v)$ for all $v \in S$. For any $T \subseteq V(G)$ with $S \subseteq T$, we say that $\phi|_T$ extends $\phi|_S$.

11.1 Upper Bound

In this section we give the upper bound for the complexity of q -CLIQUE COLORING parameterized by treewidth. The algorithm is bottom-up dynamic programming along a nice tree decomposition $(T, \mathcal{B} = \{B_t\}_{t \in V(T)})$ of the input graph G . At each bag B_t , we enumerate all colorings of $G[B_t]$ and verify for each such coloring if it can be extended to G_t such that there are no monochromatic maximal cliques that use a vertex from $V_t \setminus B_t$. Necessarily, we have to allow monochromatic maximal cliques S that are contained inside $G[B_t]$, since further up in the tree decomposition, there may be a vertex v that is complete to S . Therefore, all vertices in S may receive the same color, as long as v (or another such vertex) receives a different color. If on the other hand a monochromatic maximal clique has a vertex that has already been ‘forgotten’ at t , i.e. it is contained in $V_t \setminus B_t$, then this vertex has no neighbors in $V(G) \setminus V_t$; therefore, no vertex from $V(G) \setminus V_t$ can ‘fix’ this monochromatic maximal clique, and we can disregard the coloring at hand.

As a subroutine, we will have to be able to check at each bag B_t , if some subset $S \subseteq B_t$ contains a maximal clique in G_t . Doing this by brute force would add a multiplicative factor of roughly $2^{\text{tw}} \cdot n$ to the runtime which we cannot afford. To avoid this increase in the runtime, we use fast subset convolution¹ to build an oracle \mathbb{O}_t that, once constructed, can tell us in constant time whether or not any subset $S \subseteq B_t$ contains a maximal clique in G_t , for each node t . We give a dynamic programming algorithm that constructs such oracles for all nodes in the tree decomposition, to ensure that we can maintain a runtime that is linear in n . Since it suffices to construct this oracle once per node, this will infer only an additive factor of $2^{\text{tw}} \cdot \text{tw}^{\mathcal{O}(1)} \cdot n$ to the runtime, which does not increase the worst-case complexity for any $q \geq 2$.

Proposition 11.1. *Let G be a graph and (T, \mathcal{B}) a nice tree decomposition of G of width tw . There is an algorithm that constructs a family of oracles $\{\mathbb{O}_t\}_{t \in V(T)}$ in time $2^{\text{tw}} \cdot \text{tw}^{\mathcal{O}(1)} \cdot |V(T)|$ that, once constructed, has the following property. For every $t \in V(T)$ and $S \subseteq B_t$, \mathbb{O}_t answers in constant time whether or not S contains a maximal clique in G_t .*

¹Similar ideas have been used by Cochefert and Kratsch [78] to give an $\mathcal{O}^*(2^n)$ -time algorithm for CLIQUE COLORING.

Proof. For each $t \in V(T)$, Let $f_t: 2^{B_t} \rightarrow \{0, 1\}$ be the function defined as follows. For all $S \subseteq B_t$, we let

$$f_t(S) := \begin{cases} 1, & \text{if } S \text{ contains a maximal clique in } G_t, \\ 0, & \text{otherwise.} \end{cases}$$

To prove the statement, we have to show how to compute all values of f_t , for all $t \in V(T)$, within the claimed time bound.

As a first step, we show how to compute a family of functions $\{g_t: 2^{B_t} \rightarrow \{0, 1\}\}_{t \in V(T)}$ such that for all $t \in V(T)$ and all $S \subseteq B_t$, $g_t(S) = 1$ if and only if S is a maximal clique in G_t . We do this by bottom-up dynamic programming, and now describe how to compute the function g_t assuming that the functions at the children of t , if any, have been computed.

Leaf Node. If $t \in V(T)$ is a leaf node, then $B_t = \emptyset$, and there is nothing to compute.

Introduce Node. Suppose $t \in V(T)$ is an introduce node with child s and let v be the vertex introduced at t . Let $S \subseteq B_t$. There are two cases we have to consider, first when $v \notin S$ and second when $v \in S$. If $v \notin S$, then S is a maximal clique in G_t if and only if S is a maximal clique in G_s and S is *not* complete to v . If $v \in S$, then any clique containing S must be fully contained in B_t , since v has no neighbors in $V_t \setminus B_t$. To summarize, we set:

$$g_t(S) := \begin{cases} 1, & \text{if either } v \notin S, g_s(S) = 1, \text{ and } S \not\subseteq N(v) \\ & \quad \text{or } v \in S \text{ and } S \text{ is a maximal clique in } G[B_t] \\ 0, & \text{otherwise} \end{cases}$$

Forget Node. If $t \in V(T)$ is a forget node with child s , then we have that $G_t = G_s$. Therefore, for each $S \subseteq B_t$, it suffices to set $g_t(S) := g_s(S)$.

Join Node. Suppose $t \in V(T)$ is a join node with children s_1 and s_2 . Then, for each $S \subseteq B_t$, we have that S is a maximal clique in G_t if and only if it is both a maximal clique in G_{s_1} and in G_{s_2} . Therefore, we let $g_t(S) := g_{s_1}(S) \cdot g_{s_2}(S)$.

Computing an entry of a function at an introduce node takes time at most $\text{tw}^{\mathcal{O}(1)}$, and for a forget or join node it can be done in time $\mathcal{O}(1)$. Therefore, the family $\{g_t\}_{t \in V(T)}$ can be computed in time $2^{\text{tw}} \cdot \text{tw}^{\mathcal{O}(1)} \cdot |V(T)|$.

For the remainder of the proof, recall that for a set Ω , and two functions α and β defined on 2^Ω , their *subset convolution* \circledast is defined as: for all $S \in 2^\Omega$, $(\alpha \circledast \beta)(S) = \sum_{T \subseteq S} \alpha(T)\beta(S \setminus T)$. Fix some $t \in V(T)$. We define a constant function $c_t: 2^{B_t} \rightarrow \{1\}$, meaning that $c_t(S) = 1$ for all $S \subseteq B_t$, and construct a function $h_t := g_t \circledast c_t$. Using the algorithm of Björklund et al. [25], all values of h_t can be computed in time $2^{\text{tw}} \cdot \text{tw}^{\mathcal{O}(1)}$. By construction, each set $S \subseteq B_t$ contains $h_t(S)$ maximal cliques in G_t . We therefore obtain f_t as:

$$\forall S \subseteq B_t: f_t(S) := \begin{cases} 1, & \text{if } h_t(S) \geq 1, \\ 0, & \text{otherwise,} \end{cases}$$

Computing the family of functions $\{f_t\}_{t \in V(T)}$ and therefore the family of oracles $\{\mathbb{O}_t\}_{t \in V(T)}$ this way can be done within an additional runtime of $2^{\text{tw}} \cdot \text{tw}^{\mathcal{O}(1)} \cdot |V(T)|$. \square

We are now ready to give the algorithm. We assume that we are given a width- tw tree decomposition of the input graph whose tree has $\mathcal{O}(\text{tw} \cdot n)$ nodes. This requirement on the number of nodes is standard, see e.g. [93].

Theorem 11.2. *For any fixed $q \geq 2$, there is an algorithm that given an n -vertex graph G and a tree decomposition of G of width tw which has $\mathcal{O}(\text{tw} \cdot n)$ nodes, decides whether G has a clique coloring with q colors in time $\mathcal{O}(q^{\text{tw}} \cdot \text{tw}^{\mathcal{O}(1)} \cdot n)$, and constructs one such coloring, if it exists.*

Proof. First, we transform the given tree decomposition of G into a nice tree decomposition (T, \mathcal{B}) . This can be done in $\mathcal{O}(\text{tw}^2 \cdot n)$ time by Lemma 3.3. We may assume that the bags at leaf nodes are empty, and that T is rooted in some node $\mathbf{r} \in V(T)$, and $B_{\mathbf{r}} = \emptyset$.

We do standard bottom-up dynamic programming along T . Let $t \in V(T)$. A partial solution is a q -coloring of G_t that satisfies one additional property. Suppose that in some coloring of G_t , there is a monochromatic maximal clique X in G_t that has some vertex $v \in V_t \setminus B_t$. Then, v has no neighbors in $V(G) \setminus V_t$, therefore X is also a maximal clique in G . This means that the present coloring cannot be extended to a coloring in which X becomes non-maximal, and therefore we can disregard it.

In light of this, we define the table entries as follows. For each $t \in V(T)$ and function $\gamma_t: B_t \rightarrow \{1, \dots, q\}$, we let $\text{tab}[t, \gamma_t] = 1$ if and only if there is a q -coloring γ of G_t such that

- $\gamma|_{B_t} = \gamma_t$, and
- for each maximal clique X in G_t that is monochromatic under γ , $X \subseteq B_t$.

Since $B_{\mathbf{r}} = \emptyset$, we can immediately observe that the solution to the instance can be read off the table entries at the root node, once computed. Throughout the following we denote by γ_{\emptyset} the q -coloring defined on an empty domain.

Observation 11.2.1. G has a clique coloring with q colors if and only if $\text{tab}[\mathbf{r}, \gamma_{\emptyset}] = 1$.

As a preprocessing step, we compute the family of oracles $\{\mathbb{O}_t\}_{t \in V(T)}$ from Proposition 11.1 which will be used at forget nodes. We now show how to compute the table entries for the different types of nodes, assuming that the table entries at the children, if any, have previously been computed.

Leaf Node. If t is a leaf node, then $B_t = \emptyset$ and we only have to consider the empty coloring. We set $\text{tab}[t, \gamma_{\emptyset}] = 1$.

Introduce Node. Let $t \in V(T)$ be an introduce node with child s , and let v be the vertex introduced at t , i.e. we have that $B_t = B_s \cup \{v\}$. Since $V_t \setminus B_t =$

$(V_t \setminus \{v\}) \setminus (B_t \setminus \{v\}) = V_s \setminus B_s$, and since v has no neighbors in $V_t \setminus B_t$ by the properties of a tree decomposition, it is clear that a coloring of G_t has a monochromatic maximal clique with a vertex in $V_t \setminus B_t$ if and only if its restriction to V_s is a coloring of G_s that has a monochromatic maximal clique with a vertex in $V_s \setminus B_s$. Therefore, for each $\gamma_t: B_t \rightarrow \{1, \dots, q\}$, we simply let $\text{tab}[t, \gamma_t] = 1$ if and only if $\text{tab}[s, \gamma_t|_{B_s}] = 1$.

Join Node. Let $t \in V(T)$ be a join node with children s_1 and s_2 and recall that $B_t = B_{s_1} = B_{s_2}$. In this case, for any $\gamma_t: B_t \rightarrow \{1, \dots, q\}$, G_t has a q -coloring γ with $\gamma|_{B_t} = \gamma_t$ without a monochromatic maximal clique in $V_t \setminus B_t$ if and only if the analogous condition holds for both G_{s_1} and G_{s_2} . Therefore, for all such γ_t , we let $\text{tab}[t, \gamma_t] = 1$ if and only if $\text{tab}[s_1, \gamma_t] = \text{tab}[s_2, \gamma_t] = 1$.

Forget Node. Let $t \in V(T)$ be a forget node with child s and let v be the vertex forgotten at t , i.e. $B_s = B_t \cup \{v\}$. A partial solution at node s may have a monochromatic maximal clique using the vertex v , provided that the clique is fully contained in B_s , while partial solutions at the node t may not. On the other hand, a clique that is maximal inside B_s may not be maximal in G_s ; and as soon as a maximal clique uses a vertex from $V_s \setminus B_s$ it is not monochromatic in any partial solution at the node s , as asserted by the definition of the table entries. We can therefore consult with the oracle \mathbb{O}_s to verify if the intersection of any color class with the neighborhood of v , together with the vertex v , contains a maximal clique in G_s (and not just in $G[B_s]$). Therefore, for a given coloring $\gamma_t: B_t \rightarrow \{1, \dots, q\}$, we can check whether or not there is a partial solution in G_t whose restriction to B_t is equal to γ_t as follows. For each color $c \in \{1, \dots, q\}$, extend γ_t to a coloring γ_s of B_s by assigning vertex v color c . If $\text{tab}[s, \gamma_s] = 1$, then we check if color class c does *not* contain a maximal clique in G_s , in which case we can set $\text{tab}[t, \gamma_t]$ to 1. If there is no color c passing these checks then we know that we can set $\text{tab}[t, \gamma_t]$ to 0. We summarize this in Algorithm 7.

This completes the description of the algorithm. Correctness follows from the description of the computation of the table entries, by induction on the height of each node. For the runtime, we first take $2^{\text{tw}} \cdot \text{tw}^{\mathcal{O}(1)} \cdot n$ time to construct the clique oracles. At each node, there are at most $q^{\text{tw}+1}$ table entries to consider, and it is clear that the computation of a table entry at a leaf, introduce, or join node takes constant time. With the clique oracle at hand, computing an entry at a forget node takes time $\mathcal{O}(q) = \mathcal{O}(1)$. Therefore, at each node all table entries can be computed in time $\mathcal{O}(q^{\text{tw}})$ and since there are $\mathcal{O}(\text{tw} \cdot n)$ nodes in the tree decomposition, all table entries are computed in time $q^{\text{tw}} \cdot \text{tw}^{\mathcal{O}(1)} \cdot n$, which, since $q \geq 2$, bounds the total runtime of the algorithm. Using standard memoization techniques, the algorithm can also construct a coloring, if one exists. \square

We would like to remark that the algorithm of the previous theorem subsumes a recent linear-time algorithm for CLIQUE COLORING on outerplanar graphs. Liang et

```

Input:  $G, (T, \mathcal{B})$  as above, forget node  $t \in V(T)$ 
1 Let  $v \in B_s \setminus B_t$  be the vertex forgotten at  $t$ ;
2 Construct the clique oracle  $\mathbb{O}_t$  of  $G[B_t]$  using Proposition 11.1;
3 foreach  $\gamma_t: B_t \rightarrow \{1, \dots, q\}$  do
4    $\text{tab}[t, \gamma_t] \leftarrow 0$ ;
5   foreach  $c \in \{1, \dots, q\}$  do
6     Let  $\gamma_s: B_s \rightarrow \{1, \dots, q\}$  be such that for all  $u \in B_t$ ,  $\gamma_s(u) = \gamma_t(u)$ ,
      and  $\gamma_s(v) = c$ ;
7     if  $\text{tab}[s, \gamma_s] = 1$  then
8       Let  $S \leftarrow (N(v) \cap \gamma_t^{-1}(c)) \cup \{v\}$ ;
9       if  $\mathbb{O}_s(S) = 0$  then  $\text{tab}[t, \gamma_t] \leftarrow 1$ ;

```

Algorithm 7: Algorithm to compute all table entries at a forget node t with child s , assuming all table entries at s have been computed. (Notation: For a set $S \subseteq B_t$, $\mathbb{O}_t(S) = 0$ if and only S contains no maximal clique in G_t .)

al. [206] observed that the CLIQUE COLORING algorithm for planar graphs [195] does not construct a coloring with the optimal number of colors, and therefore they gave an algorithm on outerplanar graphs that runs in linear time and outputs a coloring. Our algorithm produces a coloring as well, and since outerplanar graphs have treewidth at most 2 [30] (see also Section 4.7.1) we can do the following. First, we compute a width-2 tree decomposition of the input graph in linear time, for instance using Bodlaender's algorithm [29], or the algorithm due to Katsikarelis [180]. In both cases, we obtain a tree decomposition with $\mathcal{O}(n)$ bags, where n is the number of vertices in the input outerplanar graph. Then we apply our algorithm whose runtime is also linear if $\text{tw} = 2$ (see Theorem 11.2). In fact, the above algorithm generalizes this to k -outerplanar graphs, for any fixed constant k , following the same reasoning (again, cf. Section 4.7.1).

11.2 Lower Bound

In this section we show that the previously presented algorithm is optimal under SETH. In fact, we prove hardness for a much larger parameter, namely the distance to a linear forest (for $q = 2$), and the distance to a caterpillar forest (for $q \geq 3$). A *linear forest* is a forest in which every connected component is a path and a *caterpillar forest* is a forest in which every connected component is a caterpillar. Note that both paths and caterpillars have pathwidth 1, and clearly, they do not contain any cycles. Therefore, a lower bound parameterized by the (vertex deletion) distance to a linear/caterpillar forest implies a lower bound for the parameter pathwidth plus feedback vertex set number. For $q = 2$, we give a reduction from *s-NOT-ALL-EQUAL SAT* (*s-NAE-SAT*) on n variables. Cygan et al. [92] showed that under SETH, for

any $\epsilon > 0$, there is some constant s such that s -NAE-SAT cannot be solved in time $\mathcal{O}^*((2 - \epsilon)^n)$. For all $q \geq 3$, we proceed as follows. First, we consider the q -LIST COLORING problem, where we are given a graph G and a list for each of its vertices which is a subset of $\{1, 2, \dots, q\}$, and the question is whether G has a proper coloring such that each vertex receives a color from its list. We show that the problem has no $\mathcal{O}^*((q - \epsilon)^t)$ -time algorithms under SETH where t is the size of a given deletion set to a linear forest. We then use q -LIST-COLORING as a starting point of a reduction to obtain the desired lower bound for $q \geq 3$. Our construction uses the fact that on triangle-free graphs, the proper colorings and the clique colorings coincide, and exploits properties of Mycielski graphs. We would like to remark that in earlier work, Marx [216] also used Mycielski graphs and their properties in hardness proofs for the CLIQUE COLORING problem.

11.2.1 The Case $q = 2$

We first give the lower bound for the case $q = 2$. We would like to remark that Kratochvíl and Tuza [195] gave a reduction from NOT-ALL-EQUAL SAT to 2-CLIQUE COLORING as well, but their reduction does not imply the fine-grained lower bound we aim for here: the resulting graph is at distance $2n$ to a disjoint union of cliques of constant size (at most s). This only rules out $\mathcal{O}^*((\sqrt{2} - \epsilon)^t)$ -time algorithms parameterized by pathwidth, and does not give any lower bound if the feedback vertex set number is another component of the parameter.

Theorem 11.3. *For any $\epsilon > 0$, 2-CLIQUE COLORING parameterized by the distance t to a linear forest cannot be solved in time $\mathcal{O}^*((2 - \epsilon)^t)$, unless SETH fails.*

Proof. We give a reduction from the well-known s -NAE-SAT problem, in which we are given a boolean CNF formula ϕ whose clauses are of size at most s , and the question is whether there is a truth assignment to the variables of ϕ , such that in each clause, at least one literal evaluates to true and at least one literal evaluates to false.

Let ϕ be a boolean CNF formula on n variables x_1, \dots, x_n with maximum clause size s . We denote by $\text{clauses}(\phi)$ the set of clauses of ϕ and by $\text{vars}(C)$ the set of variables that appear in the clause C of ϕ .

Given ϕ , we construct an instance G_ϕ for 2-CLIQUE COLORING as follows. For each variable x_i , we create a vertex v_i in G . Let $V' = \{v_1, \dots, v_n\}$. For each set S of variables, let $V_S = \{v_i \mid x_i \in S\}$. For each clause C_i of ϕ , we add the following clause gadget to G_ϕ . If C_i is monotone, add a path on four vertices to G_ϕ , the end vertices of which are a_i and b_i . Make $N(a_i) \cap V' = N(b_i) \cap V' = V_{\text{vars}(C_i)}$, and make $V_{\text{vars}(C_i)} \subset V'$ a clique. If C_i is not monotone, let $\text{pos}(C)$ (resp. $\text{neg}(C)$) denote the set of variables with positive (resp. negative) literals in C . Add a path on three vertices to G_ϕ , the end vertices of which are a_i and b_i , make $N(a_i) \cap V' = V_{\text{pos}(C)}$ and make $V_{\text{pos}(C)}$ a clique. Analogously, make $N(b_i) \cap V' = V_{\text{neg}(C)}$ and make $V_{\text{neg}(C)}$

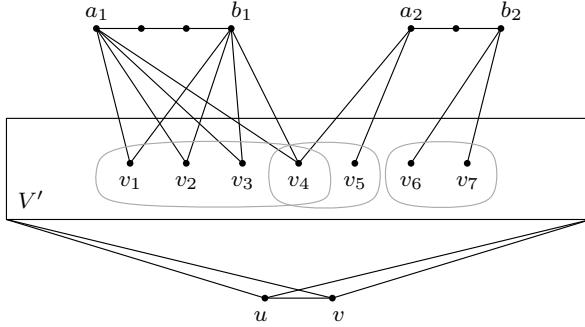


Figure 11.1: Depiction of G_ϕ with two clauses, namely a monotone clause $C_1 = \neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4$ and a non-monotone clause $C_2 = x_4 \vee x_5 \vee \neg x_6 \vee \neg x_7$. Note that $G_\phi - V'$ is a linear forest.

a clique. Finally, add two adjacent vertices u, v to G_ϕ and make $N[u] = N[v] = \{u, v\} \cup V'$. See Figure 11.1.

We will show that G_ϕ is a yes-instance to 2-CLIQUE COLORING if and only if ϕ is a yes-instance to s -NAE-SAT. We first make the following observation about the maximal cliques of G_ϕ , which follows directly from the fact that the vertices u and v are complete to V' .

Observation 11.3.1. *The vertices u and v belong to every maximal clique of $G_\phi[V' \cup \{u, v\}]$.*

Claim 11.3.2. *Let $f : V(G_\phi) \rightarrow \{0, 1\}$ be a 2-clique coloring of G_ϕ and C_i be a clause of ϕ . Then, if C_i is monotone, then $f(a_i) \neq f(b_i)$. Otherwise, $f(a_i) = f(b_i)$.*

Proof. If C_i is monotone, a_i and b_i are the end vertices of a path on four vertices, each edge of which is a maximal clique of G_ϕ . Thus, $f(a_i) \neq f(b_i)$ in any 2-clique coloring f of G_ϕ . Similarly, if C_i is not monotone, a_i and b_i are the end vertices of a path on three vertices, each edge of which is a maximal clique of G_ϕ . Hence $f(a_i) = f(b_i)$. \square

Now, suppose G has a 2-clique coloring $f : V(G_\phi) \rightarrow \{0, 1\}$. We construct a truth assignment for $\{x_1, \dots, x_n\}$ according to the colors assigned to the vertices of V' by f . That is, if $f(v_i) = 0$, we set x_i to false, and if $f(v_i) = 1$, we set x_i to true. We will now show that this assignment satisfies all clauses of ϕ . Let C_i be a clause of ϕ . First, assume that C_i is monotone. By Claim 11.3.2, $f(a_i) \neq f(b_i)$. Since $V_{\text{vars}(C_i)} \cup \{a_i\}$ is a maximal clique of G_ϕ , the vertices of $V_{\text{vars}(C_i)}$ cannot all be colored with $f(a_i)$. Similarly, $V_{\text{vars}(C_i)} \cup \{b_i\}$ is a maximal clique of G_ϕ , the vertices of $V_{\text{vars}(C_i)}$ cannot all be colored with $f(b_i)$. Thus, there exist two vertices $v_j, v_k \in V_{\text{vars}(C_i)}$ such that $f(v_j) \neq f(v_k)$. Since C_i is monotone, this implies that

x_j and x_k are not both evaluated to the same value and therefore C_i is satisfied. Now assume C_i is not monotone. By Claim 11.3.2, $f(a_i) = f(b_i)$. Hence, since $V_{\text{pos}(C_i)} \cup \{a_i\}$ and $V_{\text{neg}(C_i)} \cup \{b_i\}$ are maximal cliques of G , there exists $v_j \in V_{\text{pos}(C_i)}$ and $v_k \in V_{\text{neg}(C_i)}$ such that $f(v_j) = f(v_k)$. This implies that x_j and x_k are not evaluated to the same value under the proposed assignment and thus C_i is satisfied.

For the other direction, assume ϕ admits an assignment ξ satisfying all clauses. We construct a clique coloring $f : V(G_\phi) \rightarrow \{0, 1\}$ for G_ϕ in the following way. Color the vertices of V' according to the assignment of the variables of ϕ . That is, if $\xi(x_i) = \text{true}$ (resp. $\xi(x_i) = \text{false}$), define $f(v_i) = 1$ (resp. $f(v_i) = 0$). If C_i is monotone, let $a_i a'_i b'_i b_i$ be the path on four vertices connecting a_i and b_i in the clause gadget of C_i . Define $f(a_i) = f(b'_i) = 1$ and $f(a'_i) = f(b_i) = 0$. If C_i is not monotone, let $a_i a'_i b_i$ be the three vertex path connecting a_i and b_i in the clause gadget of C_i . If all the vertices of either $V_{\text{pos}(C_i)}$ or $V_{\text{neg}(C_i)}$ are colored 1, set $f(a_i) = f(b_i) = 0$ and $f(a'_i) = 1$. Otherwise set $f(a_i) = f(b_i) = 1$ and $f(a'_i) = 0$. Finally, define $f(u) = 0$ and $f(v) = 1$. To see that this is indeed a 2-clique coloring of G_ϕ , first note that by Observation 11.3.1, no maximal clique contained in $G_\phi[V' \cup \{u, v\}]$ is monochromatic. Furthermore, since all paths of the clause gadgets are properly colored, no maximal clique contained in $G_\phi - (V' \cup \{u, v\})$ is monochromatic. It remains to show that for each clause C_i , the maximal cliques defined by $N[a_i]$ and $N[b_i]$ are not monochromatic. Let C_i be a monotone clause. Since C_i is satisfied, there exist $x_j, x_k \in \text{vars}(C_i)$ such that $\xi(x_j) \neq \xi(x_k)$. Hence, $f(v_j) \neq f(v_k)$, which shows that $N[a_i]$ and $N[b_i]$ are each not monochromatic. If C_i is not monotone, by definition the vertices of $N[a_i]$ and $N[b_i]$ are not all colored 1. Suppose all the vertices of $N[a_i]$ are colored 0. In particular, we have $f(a_i) = f(b_i) = 0$. This implies that, by construction, all the vertices of $N(b_i) = V_{\text{neg}(C_i)}$ are colored 1. However, this is a contradiction with the fact that the clause C_i is satisfied, since all its literals are evaluated to false. Hence, f is indeed a 2-clique coloring of G_ϕ .

Finally, note that $G - V'$ is a disjoint union of paths of length at most four. Hence, G is at distance n to a linear forest. Therefore, if for some $\epsilon > 0$, 2-CLIQUE COLORING parameterized by the distance t to a linear forest can be solved in time $\mathcal{O}^*((2 - \epsilon)^t)$, then s -NAE-SAT can be solved in time $\mathcal{O}^*((2 - \epsilon)^n)$, which would contradict SETH [92]. This concludes the proof. \square

11.2.2 q -List Coloring Parameterized by Treewidth

To handle the case $q \geq 3$, we first show a lower bound for q -LIST COLORING on triangle-free graphs parameterized by treewidth. In particular, we make use of the following reduction, which strengthens the lower bound parameterized by feedback vertex set due to Lokshtanov et al. [208], to the parameterization distance to linear forest. The following lemma describes the clause gadget that will be used in the reduction.

Lemma 11.4. *For each $q \geq 3$ there is a polynomial-time algorithm that, given a*

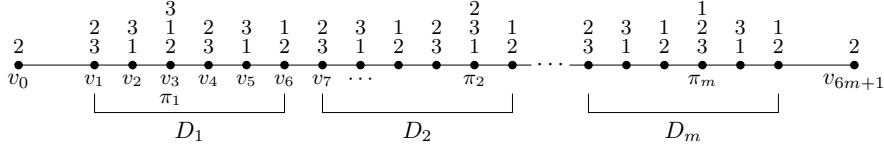


Figure 11.2: An example 3-LIST-COLORING instance created as in the proof of Lemma 11.4, where $(c_1, \dots, c_m) = (3, 2, \dots, 1)$.

vector $(c_1, \dots, c_m) \in [q]^m$, outputs a q -list-coloring instance (P, Λ) where P is a path of size $\mathcal{O}(m)$ containing distinguished vertices (π_1, \dots, π_m) , such that the following holds. For each $(d_1, \dots, d_m) \in [q]^m$ there is a proper list-coloring γ of P in which $\gamma(\pi_i) \neq d_i$ for all i , if and only if $(c_1, \dots, c_m) \neq (d_1, \dots, d_m)$.

Proof. The path P consists of consecutive vertices $v_0, v_1, \dots, v_{6m}, v_{6m+1}$. Vertex v_0 is the source and v_{6m+1} is the sink. The remaining $6m$ vertices are split into m groups D_1, \dots, D_m consisting of six consecutive vertices $v_{6(i-1)+1}, \dots, v_{6i}$ ($i \in [m]$) each. We first add some colors to the lists of these vertices which are allowed regardless of (c_1, \dots, c_m) . Later we will add some more colors to the lists of selected vertices to obtain the desired behavior.

Initialize the ‘default’ list of vertex v_i for $i \in [6m]$ to contain the two colors $\{(i \bmod 3) + 1, (i + 1 \bmod 3) + 1\}$, so that the first few lists are $\{2, 3\}$, $\{3, 1\}$, $\{1, 2\}$, and so on. Initialize $\Lambda(v_0) := \Lambda(v_{6m+1}) := \{2\}$. With these lists, there is no proper list-coloring of P . The color for the source vertex is fixed to 2, forcing the color of v_1 to 3, which forces v_2 to 1, and generally forces v_i to color $(i + 1) \bmod 3 + 1$. Hence v_{6m} is forced to $(6m + 1) \bmod 3 + 1 = 2$, creating a conflict with the sink v_{6m+1} which is also forced to color 2.

We now introduce additional colors on some lists, and identify the distinguished vertices π_1, \dots, π_m among the vertices $v_{i'}$ (where $i' \in [6m]$), to allow proper list-colorings under the stated conditions. (Note that in the rest of the proof, we will make use of two symbols for any distinguished vertex, depending on which is more convenient at the time: π_i where $i \in [m]$ and $v_{i'}$ where $i' \in [6m]$.) For a group D_i of six consecutive vertices, the *interior* of the group consists of the middle four vertices. For each index $i \in [m]$, choose π_i as a vertex from the interior of group D_i such that c_i is not on the default list of colors for π_i . Since there is no color that appears on all of the default lists of the four interior vertices, this is always possible. Add c_i to the list of allowed colors for π_i . This completes the construction of the list-coloring instance (P, Λ) . For an illustration see Figure 11.2.

It is easy to see that the construction can be performed in polynomial time. To conclude the proof, we argue that (P, Λ) has the desired properties. Observe that if $(d_1, \dots, d_m) = (c_1, \dots, c_m)$, then a proper list-coloring γ of P in which $\gamma(\pi_i) \neq d_i = c_i$ for all $i \in [m]$ would in fact be a proper list-coloring of P under the default lists before augmentation, which is impossible as we argued earlier. It remains to

argue that when (d_1, \dots, d_m) differs from (c_1, \dots, c_m) in at least one position, then P has a proper list-coloring γ with $\gamma(\pi_i) \neq d_i$ for all $i \in [m]$. To construct such a list-coloring, for each index $i \in [m]$ with $c_i \neq d_i$, assign vertex π_i the color c_i . Since the vertices π_i are interior vertices of their groups, the distinguished vertices are pairwise nonadjacent and this does not result in any conflicts. For distinguished vertices π_i with $c_i = d_i$, we will assign π_i a color from the default list of vertex π_i ; since c_i is not on the default list this results in the desired color-avoidance. We therefore conclude by verifying that the remaining vertices can be assigned a proper color from their default list.

To do so, assign the source vertex its forced color and propagate the coloring as described above, until we reach the first distinguished vertex π_i with $c_i \neq d_i$ (where $i \in [m]$). Let $i' \in [6m]$ denote the index of π_i among all vertices of P , i.e. $\pi_i = v_{i'}$. In the current partial coloring, $v_{i'-1}$ received color $((i' - 1) + 1) \bmod 3 + 1 = i' \bmod 3 + 1$ which is a color on the default list of $v_{i'}$. Hence, we do not create a conflict between vertices $v_{i'-1}$ and $v_{i'}$ as we gave $v_{i'}$ the color c_i which was not on $v_{i'}$'s default list by construction. The other color on the default list of $v_{i'}$ is $(i' + 1) \bmod 3 + 1$, which is also on the list of $v_{i'+1}$, as $\Lambda(v_{i'+1}) = \{(i' + 1) \bmod 3 + 1, (i' + 2) \bmod 3 + 1\}$. Hence, assigning $v_{i'+1}$ color $(i' + 1) \bmod 3 + 1$ does not create a conflict between $v_{i'}$ and $v_{i'+1}$, again since we assigned $v_{i'}$ a color which was not on its default list.

- If i was the last index for which $c_i \neq d_i$, then, for all $i'' \in [(i' + 2)..6m]$ we continue giving vertex $v_{i''}$ color $i'' \bmod 3 + 1$. This way the sink can be properly list-colored.
- If not, we give $v_{i'+2}$ color $i' \bmod 3 + 1 (= (i' + 3) \bmod 3 + 1)$. Note that since all distinguished vertices are interior vertices of the groups, $v_{i'+2}$ cannot be a distinguished vertex and hence has not been previously assigned a color. We now propagate this coloring along the path as before until we reach the next distinguished vertex which has already been assigned a color.

We repeat the construction until all vertices are properly list-colored. □

Theorem 11.5. *For any $\epsilon > 0$ and constant $q \geq 3$, q -LIST-COLORING on triangle-free graphs parameterized by the distance t to a linear forest cannot be solved in time $\mathcal{O}^*((q - \epsilon)^t)$, unless SETH fails.*

Proof. We show that if q -LIST-COLORING on triangle-free graphs can be solved in time $\mathcal{O}^*((q - \epsilon)^t)$ for $q \geq 3$ and some $\epsilon > 0$, then s -SAT can be solved in time $\mathcal{O}^*((2 - \delta)^n)$ for some $\delta > 0$ and any $s \in \mathcal{O}(1)$, contradicting SETH.

Suppose we have an instance φ of s -SAT on variables x_1, \dots, x_n . We construct a graph G and lists $\Lambda: V(G) \rightarrow [q]$, such that G is properly list-colorable if and only if φ is satisfiable. The first part of the reduction is inspired by the reduction of Lokshtanov et al. [208], which we repeat here for completeness. We choose an integer constant p depending on q and ϵ and group the variables of φ into t groups

F_1, \dots, F_t of size $\lfloor \log(q^p) \rfloor$ each. We call a truth assignment for the variables in F_i a *group assignment*. We say that a group assignment satisfies clause $C_j \in \varphi$ if C_j contains at least one literal which is set to **true** by the group assignment. For each group F_i , we add a set of p vertices v_i^1, \dots, v_i^p to G , in the following denoted by \mathcal{V}_i with $\Lambda(v_i^j) = [q]$ for all i and j . Each coloring of the vertices \mathcal{V}_i will encode one group assignment of F_i . We fix some efficiently computable injection $f_i: \{0, 1\}^{|F_i|} \rightarrow [q]^p$ that assigns to each group assignment for F_i a distinct p -tuple of colors. This is possible since there are $q^p \geq 2^{|F_i|}$ colorings of p vertices with q colors. For a variable $x_i \in \varphi$ we can identify the set of vertices whose colorings encode the assignment of the group containing x_i . Since each group has size $\lfloor \log(q^p) \rfloor$, the truth assignments of a variable $x_i \in \varphi$ are encoded by (some) colorings of the vertices in $\mathcal{V}_{i'}$, where $i' = \lceil i / \lfloor \log(q^p) \rfloor \rceil$.

We now construct the main part of the graph G . Let $C_j \in \varphi$ be a clause on variables $x_{j_1}, \dots, x_{j_{s'}}$, where $s' \in [s]$. The truth assignments of these variables are encoded by the colorings of the vertices in $\mathcal{V}_{C_j} := \bigcup_{i \in [s']} \mathcal{V}_{\lceil j_i / \lfloor \log(q^p) \rfloor \rceil}$. We say that a coloring $\mu: \mathcal{V}_{C_j} \rightarrow [q]$ is a *bad* coloring for C_j if there is a group for which the coloring does not represent a group assignment, or if the group assignments encoded by μ do not satisfy clause C_j .

For each bad coloring μ we construct a path using Lemma 11.4 which ensures that G is not properly list-colorable if μ appears on \mathcal{V}_{C_j} . Let $j'_i := \lceil j_i / \lfloor \log(q^p) \rfloor \rceil$ and consider the following vector of colors induced by μ :

$$c_\mu = \left(\mu(v_{j'_1}^1), \dots, \mu(v_{j'_1}^p), \dots, \mu(v_{j'_{s'}}^1), \dots, \mu(v_{j'_{s'}}^p) \right) \quad (11.1)$$

We add to G a path P_{c_μ} constructed according to Lemma 11.4 with c_μ as the input vector of colors. Let $(\pi_1, \dots, \pi_{p \cdot s'})$ denote the distinguished vertices of P_{c_μ} . We make each variable vertex $v_{j'_i}^\ell \in \mathcal{V}_{C_j}$ (where $i \in [s']$ and $\ell \in [p]$) adjacent to the distinguished vertex $\pi_{p \cdot (i-1) + \ell}$ in P_{c_μ} , intending to ensure that if all vertices in \mathcal{V}_{C_j} are colored according to μ , then this partial list-coloring on G cannot be extended to P_{c_μ} . Adding such a path for each clause in φ and each bad coloring finishes the construction of (G, Λ) .

We first count the number of vertices in G and then prove the correctness of the reduction. There are $\mathcal{O}(n)$ variable vertices and for each of the m clauses, there are at most $q^{p \cdot s}$ bad colorings, each of which adds a path on at most $\mathcal{O}(p \cdot s)$ vertices to G , by Lemma 11.4. Hence, the number of vertices in G is at most

$$\mathcal{O}(n + m \cdot q^{p \cdot s} (p \cdot s)) = \mathcal{O}(n + m) = n^{\mathcal{O}(1)}, \quad (11.2)$$

as $p, q, s \in \mathcal{O}(1)$ and $m = \mathcal{O}(n^s)$.

Claim 11.5.1. (G, Λ) is properly q -list-colorable if and only if φ has a satisfying assignment.

Proof. Suppose φ has a satisfying assignment ψ . For each group \mathcal{V}_i the assignment φ dictates a group assignment, which corresponds to a coloring on \mathcal{V} by the chosen injection f_i . Let $\gamma_\psi: \bigcup_i \mathcal{V}_i \rightarrow [q]$ denote the coloring of the variable vertices that encodes ψ . We argue that γ_ψ can be extended to the rest of G , respecting the lists Λ . For every $C_j \in \varphi$ on variables $x_{j_1}, \dots, x_{j_{s'}}$ and every bad coloring $\mu: \bigcup_{i=1}^{s'} \mathcal{V}_{j'_i} \rightarrow [q]$ w.r.t. C_j (where $j'_i = \lceil j_i / (\log(q^p)) \rceil$), we added a path P_{c_μ} to G , constructed according to Lemma 11.4, whose distinguished vertices we denote by $(\pi_1, \dots, \pi_{p \cdot s'})$. Note that c_μ denotes the vector representation of the coloring μ as in (11.1). Let c_γ denote the vector representation of γ restricted to the variable vertices $\bigcup_{i=1}^{s'} \mathcal{V}_{j'_i}$, appearing in the same order as in c_μ . Since γ_ψ encodes a satisfying assignment of φ , $c_\mu \neq c_\gamma$. Hence, by Lemma 11.4, we can extend γ_ψ to P_{c_μ} without creating a conflict; it asserts that there is a proper list-coloring γ' on P_{c_μ} such that $\gamma(v_{j'_i}^\ell) = c_\gamma(p \cdot (i-1) + \ell) \neq \gamma'(\pi_{p \cdot (i-1) + \ell})$ for all $i \in [s']$ and $\ell \in [p]$. Hence, every pair of adjacent vertices between the vertices of P_{c_μ} and the vertices encoding the truth assignments of the variables in C_j can be list-colored properly and we can conclude that γ_ψ can be extended to P_{c_μ} and subsequently, to all of G .

Now suppose (G, Λ) has a proper list-coloring γ and assume for the sake of a contradiction that φ does not have a satisfying assignment. Then, the restriction of any list-coloring of G to (some of) the variable vertices $\bigcup_i \mathcal{V}_i$ must be a bad coloring for some clause in φ . Let C_j denote such a clause for γ and let c_γ denote the corresponding vector of colors, restricted to the variable vertex groups that encode the truth assignments to the variables in C_j . We added a path P_{c_γ} to G which by Lemma 11.4 cannot be properly list-colored such that each distinguished vertex gets a color which is different from the color of the variable vertex it is adjacent to. Hence, one of the distinguished vertices of P_{c_γ} creates a conflict and we have a contradiction. \square

Observation 11.5.2. G is triangle-free and $G - \bigcup_i \mathcal{V}_i$ is a linear forest.

The previous observation can easily be verified: since G consists of the variable vertices attached to a set of disjoint paths, it is immediate that removing the variable vertices from G results in a linear forest. Moreover, the remaining edges in G are of the following form: each component of $G - \bigcup_i \mathcal{V}_i$ is connected to some subset of $\bigcup_i \mathcal{V}_i$ via a single matching. This rules out the presence of triangles in G .

By Claim 11.5.1 and Observation 11.5.2 we can now finish the proof of Theorem 11.5 in the same way as in [208].

Claim 11.5.3 (Cf. [208]). If q -LIST-COLORING on triangle-free graphs parameterized by the distance t to a linear forest can be solved in time $\mathcal{O}^*((q-\epsilon)^t)$ for some $\epsilon < 1$, then s -SAT can be solved in $\mathcal{O}^*((2-\delta)^n)$ time, for some $\delta < 1$ and any $s \in \mathcal{O}(1)$.

Proof. Let $\lambda := \log_q(q-\epsilon) < 1$, such that $(q-\epsilon)^t = q^{\lambda t}$. Note that by (11.2), the size of G is polynomial in n , the number of variables of φ . We choose a sufficiently

large p such that $\delta' = \lambda \frac{p}{p-1} < 1$. Given an instance φ of s -SAT, we use the above reduction to obtain (G, Λ) , an instance of q -LIST-COLORING, where G is a triangle-free graph. Correctness follows from Claim 11.5.1. By Observation 11.5.2 we know that G has a modulator to LINEAR FOREST of size $p \lceil \frac{n}{\lfloor p \log q \rfloor} \rceil$. By the choice of p we have $\lambda p \lceil \frac{n}{\lfloor p \log q \rfloor} \rceil \leq \lambda p \frac{n}{(p-1) \log q} + \lambda p \leq \delta' \frac{n}{\log q} + \lambda p$. Hence, s -SAT can be solved in $\mathcal{O}^*(2^{\delta'n+\lambda p}) = \mathcal{O}^*(2^{\delta'n}) = \mathcal{O}^*((2-\delta)^n)$ time for some $\delta > 0$ which does not depend on s . \square

This finishes the proof of the theorem. \square

11.2.3 The Case $q \geq 3$

We now turn to the case $q \geq 3$. Our reduction is from q -LIST-COLORING parameterized by the distance t to a linear forest, which has no $\mathcal{O}^*((q-\epsilon)^t)$ -time algorithms under SETH by Theorem 11.5.

Theorem 11.6. *For any $\epsilon > 0$ and any fixed $q \geq 3$, q -CLIQUE COLORING parameterized by the distance t to a caterpillar forest cannot be solved in time $\mathcal{O}^*((q-\epsilon)^t)$, unless SETH fails.*

Proof. We give a reduction from q -LIST COLORING on triangle-free graphs parameterized by distance to linear forest. In this proof we use the phrases “ q -colorable” as short for “can be properly colored with at most q colors”, and “ q -coloring” as short for “a proper coloring with at most q colors”. To construct our instance of q -CLIQUE COLORING, we will first describe the construction of a color selection gadget, and then describe how this gadget is attached to the rest of the graph. The description of the color selection gadget makes use of the famous Mycielski graphs. For completeness, we briefly describe how Mycielski graphs are recursively constructed and some of their useful properties. For every $p \geq 2$, the Mycielski graph M_p is a triangle-free graph with chromatic number p . For $p = 2$, we define $M_2 = K_2$. For $p \geq 3$, the graph M_p is obtained from M_{p-1} as follows. Let $V(M_{p-1}) = \{v_1, \dots, v_n\}$. Then $V(M_p) = V(M_{p-1}) \cup \{u_1, \dots, u_n, w\}$. The vertices of $V(M_{p-1})$ induce a copy of M_{p-1} in M_p , each u_i is adjacent to all the neighbors of v_i in M_{p-1} and $N(w) = \{u_1, \dots, u_n\}$. Hence, $|V(M_p)| = 3 \cdot 2^{p-2} - 1$. Moreover, it is known that M_p is edge-critical, that is, the deletion of any edge of M_p leads to a $(p-1)$ -colorable graph (see for instance [45, 209]). For our construction, we will use the graph M'_p , obtained from M_p by the deletion of an arbitrary edge xy . The following observation follows directly from the fact that M_p is edge-critical.

Observation 11.6.1. *Let M'_p be the graph obtained from M_p by the deletion of an edge xy . Then, M'_p is $(p-1)$ -colorable, and in any $(p-1)$ -coloring of M'_p , the vertices x and y receive the same color.*

Color selection gadget. We construct a gadget H_q in the following way. Consider q disjoint copies of M'_{q+1} . For $1 \leq i \leq q$, let $x_i y_i$ be the edge removed from M_{q+1} in

order to obtain the i th copy of M'_{q+1} . For each i , add $q - 1$ false twins to y_i . We denote these vertices by y_{ij} , with $1 \leq j \leq q$, $j \neq i$. Then delete the vertex y_i , for every i . Note that this graph is still q colorable and, by Observation 11.6.1, in every such q -coloring, for each i , the vertices x_i and y_{ij} , for all $j \neq i$, receive the same color. Now we add $\binom{q}{2}$ edges to connect the copies of M'_{q+1} : for $1 \leq i < j \leq q$, add the edge $y_{ij}y_{ji}$ to H_q . Note that H_q remains triangle-free after the addition of these edges, since for all $1 \leq i < j \leq q$, $N(y_{ij}) \cap N(y_{ji}) = \emptyset$. We will need the following property of the q -colorings of H_q .

Claim 11.6.2. *The graph H_q is q -colorable. Moreover, in any q -coloring ϕ of H_q , $\phi(x_i) \neq \phi(x_j)$ for all $1 \leq i < j \leq q$.*

Proof. Suppose for a contradiction that there exists a q -coloring of H_q such that $\phi(x_i) = \phi(x_j)$, for some $i \neq j$. By Observation 11.6.1, we know that $\phi(x_i) = \phi(y_{ij})$. Similarly, $\phi(x_j) = \phi(y_{ji})$. This implies that $\phi(y_{ij}) = \phi(y_{ji})$, which is a contradiction, since y_{ij} and y_{ji} are adjacent by construction. To see that a q -coloring indeed exists for H_q , first note that, by Observation 11.6.1, each copy of M'_{q+1} has a q -coloring in which x_i and y_i are assigned the same color. We can then permute the colors within a copy to obtain a proper coloring of that copy in which x_i and y_i receive color i . To complete the coloring, assign color i to every y_{ij} that is a false twin of y_i . This yields a proper q -coloring of H_q . \square

We are now ready to describe the construction of our instance G' to q -CLIQUE COLORING. Let (G, L) be an instance of q -LIST COLORING on triangle-free graphs that is at distance k from a linear forest. We construct G' as follows. Add a copy of G and a copy of H_q to G' . We denote by V' the set of vertices corresponding to $V(G)$ in G' . For each $v \in V'$, add $q - |L(v)|$ vertices adjacent to v . We denote these vertices by $\{v_j \mid j \notin L(v)\}$. Finally, make v_j adjacent to all the vertices of $\{x_\ell \mid \ell \neq j\}$. See Figure 11.3.

Note that G' is a triangle-free since H_q and G are triangle free, and $N(v_j) \cap V' = \{v\}$ and $N(v_j) \cap V(H_q)$ is an independent set. Furthermore, let $S \subseteq V(G)$ be a set such that $G - S$ is a linear forest and $|S| = t$. Then $S \cup V(H_q)$ is such that each connected component of $G' - (S \cup V(H_q))$ is a caterpillar and $|S \cup V(H_q)| = t + q(3 \cdot 2^{q-1} + q - 3) = t + \mathcal{O}(1)$, since q is a constant.

We will show that (G, L) is a yes-instance to q -LIST COLORING if and only if G' is a yes-instance to q -CLIQUE COLORING. Note that since G' is a triangle-free graph, every clique coloring of G' is actually a proper coloring of it also. First, suppose (G, L) is a yes-instance to q -LIST COLORING and let ϕ be a q -list coloring for G . We give a q -coloring ϕ' for G' in the following way. If $v \in V'$, make $\phi'(v) = \phi(v)$. For each $v_j \in N(v)$, make $\phi'(v_j) = j$. Note that since $j \notin L(v)$, we have that $\phi'(v) \neq \phi(v_j)$. Finally, consider a proper q -coloring of H_q . By Claim 11.6.2, the vertices x_1, \dots, x_q were assigned pairwise distinct colors. Without loss of generality, we can assume x_i received color i . Extend ϕ to the remaining vertices of G' according to this coloring

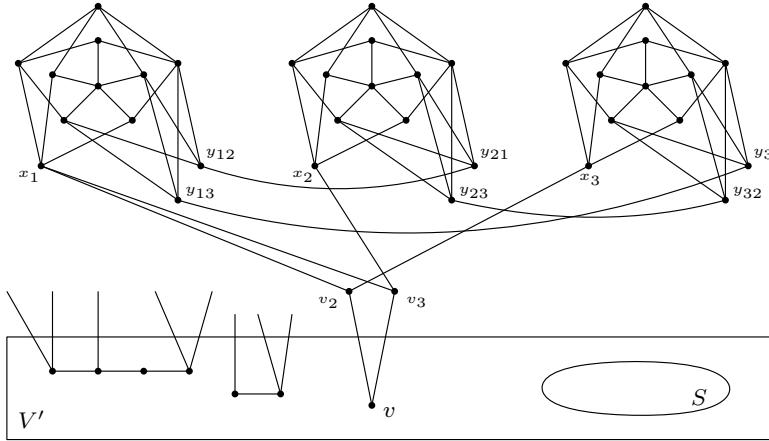


Figure 11.3: In this instance, $q = 3$ and $L(v) = \{1\}$. Note that $G' - (S \cup V(H_q))$ is a caterpillar forest.

of H_q . This leads to proper q -coloring of G' , since $\phi(v_j) = j$ and v_j is not adjacent to x_j .

Now assume G' admits a q -clique coloring ϕ . We will show that $\phi|_{V'}$ is a q -list coloring for (G, L) . Since G' is triangle-free, it is clear that $\phi|_{V'}$ is a proper coloring of G . It remains to show it satisfies the constraints imposed by the lists. By Claim 11.6.2, we can again assume that $\phi(x_i) = i$, for every i . For every $v \in V'$, since $\{x_\ell \mid \ell \neq j\} \subset N(v_j)$, we necessarily have $\phi(v_j) = j$. Finally, since for every $c \notin L(v)$ there is a neighbor of v that is colored c (namely v_c), we conclude that $\phi(v) \in L(v)$.

Now, suppose that q -CLIQUE COLORING admits an algorithm running in time $\mathcal{O}^*((q - \epsilon)^{t'})$, for some $\epsilon > 0$, where t' is the distance of the input graph to a caterpillar forest. Then, we can solve q -LIST-COLORING parameterized by the distance t to a linear forest by applying the above reduction, giving a q -CLIQUE COLORING instance at distance $t + \mathcal{O}(1)$ to a caterpillar forest, and solving the resulting q -CLIQUE COLORING instance. Correctness is argued in the previous paragraphs, and the runtime of the resulting algorithm is $\mathcal{O}^*((q - \epsilon)^{t + \mathcal{O}(1)}) = \mathcal{O}^*((q - \epsilon)^t)$, contradicting SETH by Theorem 11.5. \square

Since the instance of q -CLIQUE COLORING constructed in the proof of Theorem 11.6 is a triangle-free graph, we obtain the following corollary.

Corollary 11.7. *For any $\epsilon > 0$ and any fixed $q \geq 3$, q -COLORING on triangle-free graphs parameterized by the distance t to a caterpillar forest cannot be solved in time $\mathcal{O}^*((q - \epsilon)^t)$, unless SETH fails.*

Part IV

Further Research

Further Research

We conclude this thesis with a high-level discussion of related potential future work. Concrete open problems are stated throughout in the respective sections. The first two directions are meant to extend the impact of the ‘bounded width graph classes’ program in two different (obvious) ways. First, we can try to find more graph classes of bounded mim-width, which already has lots of algorithmic implications. In particular, we suggest to look for graph classes that have a tree-like structure rather than a linear one. The second is the search for new width measures that provide similar unification theorems. From the viewpoint of mim-width, this search can be directed both *vertically*, meaning generalizations of mim-width, or *horizontally*, i.e. width measures that are incomparable with mim-width. Naturally, a practical evaluation of this program is of great interest as well, especially due to the high generality of the associated algorithms.

Lastly, from a bird’s-eye perspective, all algorithms for vertex coloring problems on graphs of bounded clique-width follow a very similar strategy, and we believe that this can naturally be captured in a common meta-algorithm, extending such results to other variants of vertex coloring. It would be interesting to see if there is a strategy to provide matching ETH-based lower bounds for such a meta-algorithm as well.

Mim-Width: Mind the Tree

Most of the known graph classes of bounded mim-width have a linear structure, and therefore bounded *linear* mim-width, see Figure 11.4. It would be interesting to identify more graph classes where the tree structure of the decomposition is important. Concretely, to find graph classes whose linear mim-width is unbounded, while their mim-width remains bounded. Possibly we could consider the following *reverse engineering* approach to obtain such classes: take a graph class of bounded linear mim-width, and try to determine if the linear nature of the definition can be generalized to a ‘tree-like’ nature. In the process, the mim-width of the class may remain bounded while the linear mim-width may not. For instance, this can be observed when going from interval graphs to rooted directed path graphs. The real line of an interval model can be viewed as a (directed) path, and the intervals as

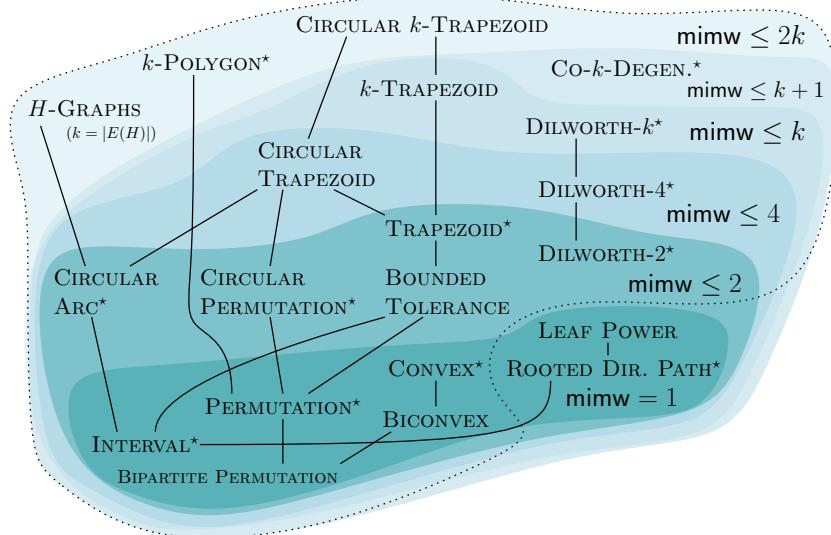


Figure 11.4: Graph classes that are known to have bounded mim-width (and unbounded clique-width). All graph classes within the dotted circle, i.e. all but LEAF POWER and ROOTED DIRECTED PATH, in fact have bounded *linear* mim-width.

(directed) subpaths. In the definition of rooted directed path graphs, the real line is replaced by a rooted tree, and the intervals by directed paths in that tree. This indeed results in a graph class of mim-width 1 whose linear mim-width is unbounded (simply because it contains all forests). Possibly, in other cases, we may even be able to identify new interesting graph classes this way. For a starting point, we believe that bounded tolerance graphs should be generalizable in a similar manner, resulting in a subclass of graphs with a so-called (h, s, t) -representation² [171]. However, for tolerance graphs we first have to give a construction of a branch decomposition of bounded linear mim-width that uses the bounded tolerance model directly; currently, such a decomposition is obtained via the construction for trapezoid graphs.

Beyond/Besides Mim-Width

Another viable research direction is the search for other width measures that unify algorithms on graph classes. Kang et al. [176] introduced *sim-width*, which strictly generalizes mim-width and is bounded on chordal and co-comparability graphs. On the downside, so far there is no problem that is known to be efficiently solvable on

²An (h, s, t) -representation of a graph G is a pair (T, \mathcal{S}) of a tree T of maximum degree h and a set $\mathcal{S} = \{S_v\}_{v \in V(G)}$ of subtrees of T of maximum degree at most s such that for all $u, v \in V(G)$, $uv \in E(G)$ if and only if $|V(S_u) \cap V(S_v)| \geq t$.

graphs of small sim-width.

Alternatively, one could look for width measures that are incomparable with mim-width. One such route could go via the *diameter* of graphs. Split graphs and co-bipartite graphs are both of unbounded mim-width, but of small diameter. On the other hand, there are plenty of graphs of large diameter and bounded mim-width. Or one could simply search for sets of graph classes on which the complexity of algorithmic problems is incomparable to those of bounded mim-width, and try to identify some common structure that allows for designing bounded width decompositions (of any kind). Even a width measure that unifies algorithmic results for a *single* problem on several graph classes could already be an interesting starting point.

Width Measures vs Graph Classes in Practice

In this thesis, we considered XP-time algorithms for graphs of bounded mim-width mainly as theoretical classification tools for polynomial-time solvability of problems on graph classes, or as a means of unifying known algorithms. Nevertheless, there is plenty of obvious motivation to bring these algorithms to practice. For instance, instead of implementing one algorithm per combination of a (σ, ρ) -problem and a graph class of bounded mim-width, it suffices to implement a single algorithm [62]. If made practical, such implementations have the potential to bring algorithmic tools to users who are not experts in structural graph theory.

First such implementations, including several methods to compute decompositions, already exist [57, 267]. For linear mim-width, it has been observed [57] that in practice these algorithms often seem to perform better than their theoretical bounds predict. Now, in line with the narrative of this thesis, it would be interesting to directly compare mim-width based algorithms with their counterparts on specific graph classes. A first study could be a comparison of an algorithm for MAXIMUM INDEPENDENT SET on interval graphs that does dynamic programming over the intersection model directly with a MAXIMUM INDEPENDENT SET algorithm on linear mim-width 1. This approach works for many types of intersection graph classes, as there are efficient methods to construct optimal decompositions from an intersection model, see the classes marked with a ‘ \star ’ in Figure 11.4. This provides an ideal setup for experiments that could tell us about the feasibility of such a program.

A Meta-algorithm for Coloring Problems on Clique-Width

The algorithms for GRAPH COLORING, b -COLORING, and CLIQUE COLORING on graphs of bounded module-width³ w all have more or less the same structure:

1. Describe the color classes of the respective colorings in terms of *types* such that the number of types depends only on the number of equivalence classes at each

³Recall: you may equivalently read ‘clique-width’, see Theorem 3.14.

node, i.e. it is bounded by some $f(w)$. (In GRAPH COLORING: the subset of equivalence classes containing a vertex of the color class.)

2. (Possibly) identify *compatibility* criteria for a pair of types τ_1, τ_2 that describes concisely the ‘merging behavior’ of color classes at internal nodes of the branch decomposition. That is, if t is a node with children r and s , then (τ_1, τ_2) is *compatible* if the union of a color class of type τ_1 in G_r and a color class of type τ_2 in G_s gives a color class in G_t . (In GRAPH COLORING, compatibility simply ensures that no edges are added between vertices of color classes with compatible types.)
3. Describe colorings simply by remembering for each type, how many color classes of that type there are. There are at most $n^{f(w)}$ many such descriptions of colorings, and this suffices to solve the instance at hand.
4. Using an auxiliary bipartite graph (the merge skeleton as it is called in Chapter 10), we can then show how to produce colorings for an internal node from the descriptions of colorings of its children, taking the notion of compatibility into account.

This overall outline can be captured in a common meta-algorithm, which is probably helpful in solving further variants of vertex-coloring problems on graphs of bounded clique-width. In some sense, this feels like the ‘natural’ approach to vertex coloring problems on graphs of bounded clique-width, and possibly, the complexity of a coloring problem may even be captured in terms of how compactly one can describe the types. If that is the case, it is even conceivable that lower bound constructions can be based on the (im)possibility (based on assumptions such as ETH) of the description of the color classes with a certain number of types.

Known ETH-based lower bounds for GRAPH COLORING and b -COLORING rule out even a subexponential dependence on the clique-width in the degree of the polynomial of the runtime, i.e. an $n^{2^{o(w)}}$ -time algorithm would violate ETH. Lower bounds of this form are rather rare and therefore particularly desirable.

Bibliography

- [1] Pierre Aboulker, Édouard Bonnet, Eun Jung Kim, and Florian Sikora. Grundy coloring & friends, half-graphs, bicliques. In Christophe Paul and Markus Bläser, editors, *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, volume 154 of *LIPICS*, pages 58:1–58:18. Schloss Dagstuhl, 2020.
- [2] Geir Agnarsson, Peter Damaschke, and Magnús M. Halldórsson. Powers of geometric intersection graphs and dispersion algorithms. *Discrete Applied Mathematics*, 132(1-3):3–16, 2003.
- [3] Ernst Althaus and Sarah Ziegler. Optimal tree decompositions revisited: A simpler linear-time FPT algorithm. *CoRR*, abs/1912.09144, 2019.
- [4] Eyal Amir. Approximation algorithms for treewidth. *Algorithmica*, 56(4):448–479, 2010.
- [5] Thomas Andreae, Martin Schughart, and Zsolt Tuza. Clique-transversal sets of line graphs and complements of line graphs. *Discrete Mathematics*, 88(1):11–20, 1991.
- [6] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k -tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.
- [7] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.
- [8] Stefan Arnborg and Andrzej Proskurowski. Characterization and recognition of partial 3-trees. *SIAM Journal on Algebraic Discrete Methods*, 7(2):305–314, 1986.
- [9] Gábor Bacsó, Sylvain Gravier, András Gyárfás, Myriam Preissmann, and András Sebo. Coloring the maximal cliques of graphs. *SIAM Journal on Discrete Mathematics*, 17(3):361–376, 2004.

- [10] Gábor Bacsó, Dániel Marx, and Zsolt Tuza. H-free graphs, independent sets, and subexponential-time algorithms. In Jiong Guo and Danny Hermelin, editors, *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *LIPICS*, pages 3:1–3:12. Schloss Dagstuhl, 2017.
- [11] Gábor Bacsó and Zsolt Tuza. Clique-transversal sets and weak 2-colorings in graphs of small maximum degree. *Discrete Mathematics and Theoretical Computer Science*, 11(2):15–24, 2009.
- [12] Vineet Bafna, Babu O. Narayanan, and R. Ravi. Nonoverlapping local alignments (weighted independent sets of axis-parallel rectangles). *Discrete Applied Mathematics*, 71(1-3):41–53, 1996.
- [13] Hans-Jürgen Bandelt and Henry Martin Mulder. Distance-hereditary graphs. *Journal of Combinatorial Theory, Series B*, 41:182–208, 1986.
- [14] Dominique Barth, Johanne Cohen, and Taoufik Faik. On the b-continuity property of graphs. *Discrete Applied Mathematics*, 155(13):1761–1768, 2007.
- [15] Rémy Belmonte, Petr A. Golovach, Pinar Heggernes, Pim van ’t Hof, Marcin Kamiński, and Daniël Paulusma. Detecting fixed patterns in chordal graphs in polynomial time. *Algorithmica*, 69(3):501–521, 2014.
- [16] Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, and Yota Otachi. Grundy distinguishes treewidth from pathwidth. To appear at ESA 2020.
- [17] Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoretical Computer Science*, 511:54–65, 2013.
- [18] Claude Berge. Färbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind. *Wissenschaftliche Zeitschrift der Martin Luther Universität Halle-Wittenberg*, 10:114, 1961.
- [19] Benjamin Bergougnoux. *Matrix decompositions and algorithmic applications to (hyper)graphs*. PhD thesis, University of Clermont Auvergne, Clermont-Ferrand, France, 2019.
- [20] Benjamin Bergougnoux. personal communication, 2020.
- [21] Benjamin Bergougnoux and Mamadou Moustapha Kanté. More applications of the d-neighbor equivalence: Connectivity and acyclicity constraints. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *Proceedings of the 27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *LIPICS*, pages 17:1–17:14. Schloss Dagstuhl, 2019.

- [22] Benjamin Bergougnoux, Charis Papadopoulos, and Jan Arne Telle. Node multiway cut and subset feedback vertex set on graphs of bounded mim-width. *CoRR*, abs/1910.00887, 2019. To appear at WG 2020.
- [23] Umberto Bertele and Francesco Brioschi. On non-serial dynamic programming. *Journal of Combinatorial Theory, Series A*, 14(2):137–148, 1973.
- [24] Daniel Binkele-Raible, Ljiljana Brankovic, Marek Cygan, Henning Fernau, Joachim Kneis, Dieter Kratsch, Alexander Langer, Mathieu Liedloff, Marcin Pilipczuk, Peter Rossmanith, and Jakub Onufry Wojtaszczyk. Breaking the 2^n -barrier for irredundance: Two lines of attack. *Journal of Discrete Algorithms*, 9(3):214–230, 2011.
- [25] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC 2007)*, pages 67–74. ACM, 2007.
- [26] Jean R. S. Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In *Graph Theory and Sparse Matrix Computation*, pages 1–29. Springer, 1993.
- [27] Hans L. Bodlaender. Classes of graphs with bounded tree-width. Technical Report RUU-CS-86-22, Department of Computer Science, Utrecht University, 1986.
- [28] Hans L. Bodlaender. On disjoint cycles. *International Journal on Foundations of Computer Science*, 5(1):59–68, 1994.
- [29] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.
- [30] Hans L. Bodlaender. A partial k -arboretum for graphs of bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998.
- [31] Hans L. Bodlaender, Nick Brettell, Matthew Johnson, Giacomo Paesani, Daniël Paulusma, and Erik Jan van Leeuwen. Steiner trees for hereditary graph classes: a treewidth perspective. *CoRR*, abs/2004.07492, 2020.
- [32] Hans L. Bodlaender, Leizhen Cai, Jianer Chen, Michael R. Fellows, Jan Arne Telle, and Dániel Marx. Open problems in parameterized and exact computation – IWPEC 2006. Technical Report UU-CS-2006-052, Department of Information and Computing Sciences, Utrecht University, 2006.
- [33] Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015.

- [34] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michał Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016.
- [35] Hans L. Bodlaender, Michael R. Fellows, and Dimitrios M. Thilikos. Derivation of algorithms for cutwidth and related graph layout parameters. *Journal of Computer and System Sciences*, 75(4):231–244, 2009.
- [36] Hans L. Bodlaender, Jens Gustedt, and Jan Arne Telle. Linear-time register allocation for a fixed number of registers. In Howard J. Karloff, editor, *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1998)*, pages 574–583. ACM/SIAM, 1998.
- [37] Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996.
- [38] Hans L. Bodlaender, Ton Kloks, Dieter Kratsch, and Haiko Müller. Treewidth and minimum fill-in on d-trapezoid graphs. *Journal of Graph Algorithms and Applications*, 2(5):1–23, 1998.
- [39] Hans L. Bodlaender and Dimitrios M. Thilikos. Constructive linear time algorithms for branchwidth. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP 1997)*, volume 1256 of *LNCS*, pages 627–637. Springer, 1997.
- [40] Hans L. Bodlaender and Dimitrios M. Thilikos. Computing small search numbers in linear time. In Rodney G. Downey, Michael R. Fellows, and Frank K. H. A. Dehne, editors, *Proceedings of the First International Workshop on Parameterized and Exact Computation (IWPEC 2004)*, volume 3162 of *LNCS*, pages 37–48. Springer, 2004.
- [41] Kenneth P. Bogart, Peter C. Fishburn, Garth Isaak, and Larry Langley. Proper and unit tolerance graphs. *Discrete Applied Mathematics*, 60(1-3):99–117, 1995.
- [42] Mikołaj Bojańczyk, Martin Grohe, and Michał Pilipczuk. Definable decompositions for graphs of bounded linear cliquewidth. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 135–144. ACM, 2018.
- [43] Mikołaj Bojańczyk and Michał Pilipczuk. Definability equals recognizability for graphs of bounded treewidth. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2016)*, pages 407–416. ACM, 2016.

- [44] Mikołaj Bojańczyk and Michał Pilipczuk. Optimizing tree decompositions in MSO. In Heribert Vollmer and Brigitte Vallée, editors, *Proceedings of the 34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, volume 66 of *LIPICS*, pages 15:1–15:13. Schloss Dagstuhl, 2017.
- [45] J. Adrian Bondy and Uppaluri S. R. Murty. *Graph Theory*, volume 244 of *Graduate Texts in Mathematics*. Springer, 2008.
- [46] Flavia Bonomo, Guillermo Durán, Frederic Maffray, Javier Marenco, and Mario Valencia-Pabon. On the b-coloring of cographs and P_4 -sparse graphs. *Graphs and Combinatorics*, 25(2):153–167, 2009.
- [47] Flavia Bonomo, Oliver Schaudt, Maya Stein, and Mario Valencia-Pabon. b-Coloring is NP-hard on co-bipartite graphs and polytime solvable on tree-cographs. *Algorithmica*, 73(2):289–305, 2015.
- [48] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.
- [49] Richard B. Borie, R. Gary Parker, and Craig A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(1-6):555–581, 1992.
- [50] Andreas Brandstädt. On leaf powers. Technical report, University of Rostock, 2010.
- [51] Andreas Brandstädt, Christian Hundt, Federico Mancini, and Peter Wagner. Rooted directed path graphs are leaf powers. *Discrete Mathematics*, 310(4):897–910, 2010.
- [52] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey*. SIAM, 1999.
- [53] Andreas Brandstädt and Vadim V. Lozin. On the linear structure and clique-width of bipartite permutation graphs. *Ars Combinatoria*, 67, 2003.
- [54] Johann Brault-Baron, Florent Capelli, and Stefan Mengel. Understanding model counting for beta-acyclic CNF-formulas. In Ernst W. Mayr and Nicolas Ollinger, editors, *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *LIPICS*, pages 143–156. Schloss Dagstuhl, 2015.
- [55] Nick Brettell, Jake Horsfield, Andrea Munaro, Giacomo Paesani, and Daniël Paulusma. Bounding the mim-width of hereditary graph classes. *CoRR*, abs/2004.05018, 2020.

- [56] Nick Brettell, Jake Horsfield, and Daniël Paulusma. Colouring $(sP_1 + P_5)$ -free graphs: a mim-width perspective. *CoRR*, abs/2004.05022, 2020.
- [57] Chiel B. ten Brinke, Frank J. P. van Houten, and Hans L. Bodlaender. Practical algorithms for linear boolean-width. In Thore Husfeldt and Iyad A. Kanj, editors, *Proceedings of the 10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*, volume 43 of *LIPICS*, pages 187–198. Schloss Dagstuhl, 2015.
- [58] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.
- [59] Binh-Minh Bui-Xuan, Ondřej Suchý, Jan Arne Telle, and Martin Vatshelle. Feedback vertex set on graphs of low clique-width. *European Journal of Combinatorics*, 34(3):666–679, 2013.
- [60] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. H-join decomposable graphs and algorithms with runtime single exponential in rankwidth. *Discrete Applied Mathematics*, 158(7):809–819, 2010.
- [61] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Boolean-width of graphs. *Theoretical Computer Science*, 412(39):5187–5204, 2011.
- [62] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoretical Computer Science*, 511:66–76, 2013.
- [63] Peter Buneman. A characterisation of rigid circuit graphs. *Discrete Mathematics*, 9(3):205–212, 1974.
- [64] Tiziana Calamoneri and Blerina Sinaimeri. Pairwise compatibility graphs: A survey. *SIAM Review*, 58(3):445–460, 2016.
- [65] C. N. Campos, Simone Dantas, and Célia Picinin de Mello. Colouring clique-hypergraphs of circulant graphs. *Electronic Notes in Discrete Mathematics*, 30:189–194, 2008.
- [66] Victor A. Campos, Carlos V. Lima, Nicolas A. Martins, Leonardo Sampaio, Marcio C. Santos, and Ana Silva. The b-chromatic index of graphs. *Discrete Mathematics*, 338(11):2072–2079, 2015.
- [67] Victor A. Campos, Carlos Vinicius G. C. Lima, and Ana Silva. Graphs of girth at least 7 have high b-chromatic number. *European Journal of Combinatorics*, 48:154–164, 2015.

- [68] Victor A. Campos, Cláudia Linhares-Sales, Rudini Sampaio, and Ana Karolinna Maia. Maximization coloring problems on graphs with few P_4 . *Discrete Applied Mathematics*, 164:539–546, 2014.
- [69] Victor A. Campos and Ana Silva. Edge-b-coloring trees. *Algorithmica*, 80(1):104–115, 2018.
- [70] Márcia R. Cerioli and André L. Korenchendler. Clique-coloring circular-arc graphs. *Electronic Notes in Discrete Mathematics*, 35:287–292, 2009.
- [71] Maw-Shang Chang and Ming-Tat Ko. The 3-Steiner root problem. In Andreas Brandstädt, Dieter Kratsch, and Haiko Müller, editors, *Proceedings of the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2007)*, volume 4769 of *LNCS*, pages 109–120. Springer, 2007.
- [72] Steven Chaplick, Martin Toepfer, Jan Voborník, and Peter Zeman. On H-topological intersection graphs. In Hans L. Bodlaender and Gerhard J. Woeginger, editors, *Proceedings of the 43rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2017)*, volume 10520 of *LNCS*, pages 167–179. Springer, 2017.
- [73] Pierre Charbit, Irena Penev, Stéphan Thomassé, and Nicolas Trotignon. Perfect graphs of arbitrarily large clique-chromatic number. *Journal of Combinatorial Theory, Series B*, 116:456–464, 2016.
- [74] Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *Journal of Computer and System Sciences*, 74(7):1188–1198, 2008.
- [75] Maria Chudnovsky and Irene Lo. Decomposing and clique-coloring (diamond, odd-hole)-free graphs. *Journal of Graph Theory*, 86(1):5–41, 2017.
- [76] Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of Mathematics*, pages 51–229, 2006.
- [77] Václav Chvátal and Peter L. Hammer. Set-packing and threshold graphs. Technical Report 73-21, Computer Science Department, University of Waterloo, Canada, 1973.
- [78] Manfred Cochefert and Dieter Kratsch. Exact algorithms to clique-colour graphs. In Viliam Geffert, Bart Preneel, Branislav Rovan, Julius Štuller, and A Min Tjoa, editors, *Proceedings of the 40th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2014)*, volume 8327 of *LNCS*, pages 187–198. Springer, 2014.
- [79] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

- [80] Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce Reed, and Udi Rotics. Polynomial-time recognition of clique-width ≤ 3 graphs. *Discrete Applied Mathematics*, 160(6):834–865, 2012. Extended abstract at LATIN 2000.
- [81] Derek G. Corneil, Helmut Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3(3):163–174, 1981.
- [82] Derek G. Corneil, Yehoshua Perl, and Lorna K. Stewart. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985.
- [83] Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM Journal on Computing*, 34(4):825–847, 2005.
- [84] David Coudert, Guillaume Ducoffe, and Alexandru Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. *ACM Transactions on Algorithms*, 15(3):1–57, 2019.
- [85] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.
- [86] Bruno Courcelle. Clique-width of countable graphs: a compactness property. *Discrete Mathematics*, 276(1-3):127–148, 2004.
- [87] Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218–270, 1993.
- [88] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.
- [89] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.
- [90] Bruno Courcelle and Sang-il Oum. Vertex-minors, monadic second-order logic, and a conjecture by Seese. *Journal of Combinatorial Theory, Series B*, 97(1):91–126, 2007.
- [91] Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *CoRR*, abs/2001.06867, 2020.
- [92] Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Transactions on Algorithms*, 12(3):41:1–41:24, 2016.

- [93] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- [94] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In Rafail Ostrovsky, editor, *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2011)*, pages 150–159. IEEE, 2011.
- [95] Konrad Dabrowski and Daniël Paulusma. Clique-width of graph classes defined by two forbidden induced subgraphs. *The Computer Journal*, 59(5):650–666, 2016.
- [96] Konrad K. Dabrowski, Mathew Johnson, and Daniël Paulusma. *Clique-width for hereditary graph classes*, volume 456 of *London Mathematical Society Lecture Note Series*, page 1–56. Cambridge University Press, 2019.
- [97] Konrad K. Dabrowski, Tomáš Masařík, Jana Novotná, Daniël Paulusma, and Paweł Rzążewski. Clique-width: Harnessing the power of atoms. *CoRR*, abs/2006.03578, 2020. To appear at WG 2020.
- [98] David Défossez. Clique-coloring some classes of odd-hole-free graphs. *Journal of Graph Theory*, 53(3):233–249, 2006.
- [99] Frank K. H. A. Dehne, Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Kim Stevens. An $\mathcal{O}(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. In Lusheng Wang, editor, *Proceedings of the 11th Annual International Conference on Computing and Combinatorics (COCOON 2005)*, volume 3595 of *LNCS*, pages 859–869. Springer, 2005.
- [100] Jitender S. Deogun and George Steiner. Polynomial algorithms for Hamiltonian cycle in cocomparability graphs. *SIAM Journal on Computing*, 23(3):520–552, 1994.
- [101] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg, fifth edition, 2016.
- [102] Gabriel Andrew Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25(1-2):71–76, 1961.
- [103] Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.
- [104] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

- [105] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- [106] Dwight Duffus, Bill Sands, Norbert Sauer, and Robert E. Woodrow. Two-colouring all two-element maximal antichains. *Journal of Combinatorial Theory, Series A*, 57(1):109–116, 1991.
- [107] Jean E. Dunbar, Sandra M. Hedetniemi, S. T. Hedetniemi, David P. Jacobs, J. Knisely, R. C. Laskar, and Douglas F. Rall. Fall colorings of graphs. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 33:257–274, 2000.
- [108] Brice Effantin, Nicolas Gastineau, and Olivier Togni. A characterization of b-chromatic and partial grundy numbers by induced subgraphs. *Discrete Mathematics*, 339(8):2157–2167, 2016.
- [109] Khaled Elbassioni, Rajiv Raman, Saurabh Ray, and René Sitters. On the approximability of the maximum feasible subsystem problem with 0/1-coefficients. In Claire Mathieu, editor, *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2009)*, pages 1210–1219. ACM/SIAM, 2009.
- [110] Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98(1):21–51, 1961.
- [111] Ehab S. Elmallah and Lorna K. Stewart. Independence and domination in polygon graphs. *Discrete Applied Mathematics*, 44(1-3):65–77, 1993.
- [112] Ehab S. Elmallah and Lorna K. Stewart. Polygon graph recognition. *Journal of Algorithms*, 26(1):101–140, 1998.
- [113] David Eppstein and Elham Havvaei. Parameterized leaf power recognition via embedding into graph products. In Christophe Paul and Michał Pilipczuk, editors, *Proceedings of the 13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, volume 115 of *LIPICS*, pages 16:1–16:14. Schloss Dagstuhl, 2019.
- [114] Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In Andreas Brandstädt and Van Bang Le, editors, *Proceedings of the 27th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2001)*, volume 2204 of *LNCS*, pages 117–128. Springer, 2001.
- [115] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741.

- [116] Taoufik Faik. About the b -continuity of graphs (extended abstract). *Electronic Notes in Discrete Mathematics*, 17:151–156, 2004.
- [117] Martin Farber. Characterizations of strongly chordal graphs. *Discrete Mathematics*, 43(2-3):173–189, 1983.
- [118] Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629–657, 2008.
- [119] Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009.
- [120] Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is NP-complete. *SIAM Journal on Discrete Mathematics*, 23(2):909–939, 2009.
- [121] Stefan Felsner, Vijay Raghavan, and Jeremy Spinrad. Recognition algorithms for orders of small width and graphs of small dilworth number. *Order*, 20(4):351–364, 2003.
- [122] Paola Festa, Panos M. Pardalos, and Mauricio G. C. Resende. Feedback set problems. In *Handbook of combinatorial optimization*, pages 209–258. Springer, 1999.
- [123] Jirí Fiala, Marcin Kamiński, Bernard Lidický, and Daniël Paulusma. The k -in-a-path problem for claw-free graphs. *Algorithmica*, 62(1-2):499–519, 2012.
- [124] Stéphane Földes and Peter Ladislav Hammer. Split graphs. In *Proceedings of the 8th Southeastern Conference on Combinatorics, Graph Theory and Computing*, volume 19 of *Congressus Numerantium*, 1977.
- [125] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010.
- [126] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: Hamiltonian cycle and the odd case of graph coloring. *ACM Transactions on Algorithms*, 15(1):9:1–9:27, 2019.
- [127] Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the tractability of optimization problems on H-graphs. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *Proceedings of the 26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *LIPICS*, pages 30:1–30:14. Schloss Dagstuhl, 2018.

- [128] Fedor V. Fomin, Dieter Kratsch, Ioan Todinca, and Yngve Villanger. Exact algorithms for treewidth and minimum fill-in. *SIAM Journal on Computing*, 38(3):1058–1079, 2008.
- [129] Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM Journal on Computing*, 44(1):54–87, 2015.
- [130] Fedor V. Fomin and Yngve Villanger. Treewidth computation and extremal combinatorics. *Combinatorica*, 32(3):289–308, 2012.
- [131] Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, 130(1-3):3–31, 2004.
- [132] Delbert Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.
- [133] Martin Fürer. Faster computation of path-width. In Veli Mäkinen, Simon J. Puglisi, and Leena Salmela, editors, *Proceedings of the 27th International Workshop on Combinatorial Algorithms (IWOCA 2016)*, volume 9843 of *LNCS*, pages 385–396. Springer, 2016.
- [134] Esther Galby, Andrea Munaro, and Bernard Ries. Semitotal domination: New hardness results and a polynomial-time algorithm for graphs of bounded mim-width. *Theoretical Computer Science*, 814:28–48, 2020.
- [135] François Le Gall. Powers of tensors and fast matrix multiplication. In Katsuhiko Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC 2014)*, pages 296–303. ACM, 2014.
- [136] Michael R. Garey, David S. Johnson, Gary L. Miller, and Christos H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic Discrete Methods*, 1(2):216–227, 1980.
- [137] Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974.
- [138] Fănică Gavril. A recognition algorithm for the intersection graphs of directed paths in directed trees. *Discrete Mathematics*, 13(3):237–249, 1975.
- [139] Fănică Gavril. Algorithms for maximum weight induced paths. *Information Processing Letters*, 81(4):203–208, 2002.
- [140] James F. Geelen, A. M. H. Gerards, and Geoff Whittle. Branch-width and well-quasi-ordering in matroids and graphs. *Journal of Combinatorial Theory, Series B*, 84(2):270–290, 2002.

- [141] Michael U. Gerber and Daniel Kobler. Algorithms for vertex-partitioning problems on graphs with fixed clique-width. *Theoretical Computer Science*, 299(1-3):719–734, 2003.
- [142] Wayne Goddard and Michael A. Henning. Independent domination in graphs: A survey and recent results. *Discrete Mathematics*, 313(7):839–854, 2013.
- [143] Petr A. Golovach and Jan Kratochvíl. Computational complexity of generalized domination: A complete dichotomy for chordal graphs. In Andreas Brandstädt, Dieter Kratsch, and Haiko Müller, editors, *Proceedings of the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2007)*, volume 4769 of *LNCS*, pages 1–11. Springer, 2007.
- [144] Petr A. Golovach, Daniël Paulusma, and Erik Jan van Leeuwen. Induced disjoint paths in AT-free graphs. In Fedor V. Fomin and Petteri Kaski, editors, *Proceedings of the 13th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2012)*, volume 7357 of *LNCS*, pages 153–164. Springer, 2012.
- [145] Petr A. Golovach, Daniël Paulusma, and Erik Jan van Leeuwen. Induced disjoint paths in circular-arc graphs in linear time. *Theoretical Computer Science*, 640:70–83, 2016.
- [146] Martin Charles Golumbic. The complexity of comparability graph recognition and coloring. *Computing*, 18(3):199–208, 1977.
- [147] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Elsevier, 1980.
- [148] Martin Charles Golumbic and Udi Rotics. On the clique-width of some perfect graph classes. *International Journal of Foundations of Computer Science*, 11(3):423–443, 2000.
- [149] Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *Journal of Computer and System Sciences*, 72(8):1386–1396, 2006.
- [150] Frank Gurski and Egon Wanke. The tree-width of clique-width bounded graphs without $K_{n,n}$. In Ulrik Brandes and Dorothea Wagner, editors, *Proceedings of the 26th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2000)*, volume 1928 of *LNCS*, pages 196–205. Springer, 2000.
- [151] Frank Gurski and Egon Wanke. The NLC-width and clique-width for powers of graphs of bounded tree-width. *Discrete Applied Mathematics*, 157(4):583–595, 2009.

- [152] Michel Habib, Ross M. McConnell, Christophe Paul, and Laurent Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1-2):59–84, 2000.
- [153] Rudolf Halin. S-functions for graphs. *Journal of Geometry*, 8(1-2):171–186, 1976.
- [154] Peter L. Hammer and Bruno Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981.
- [155] Frédéric Havet, Claudia Linhares Sales, and Leonardo Sampaio. b-coloring of tight graphs. *Discrete Applied Mathematics*, 160(18):2709–2715, 2012.
- [156] Frédéric Havet and Leonardo Sampaio. On the Grundy and b -chromatic numbers of a graph. *Algorithmica*, 65:885–899, 2013.
- [157] Pinar Heggernes and Dieter Kratsch. Linear-time certifying recognition algorithms and forbidden induced subgraphs. *Nordic Journal of Computing*, 14(1-2):87–108, 2007.
- [158] Pinar Heggernes and Jan Arne Telle. Partitioning graphs into generalized dominating sets. *Nordic Journal of Computing*, 5(2):128–142, 1998.
- [159] Petr Hliněný. Branch-width, parse trees, and monadic second-order logic for matroids. *Journal of Combinatorial Theory, Series B*, 96(3):325–351, 2006.
- [160] Petr Hliněný and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM Journal on Computing*, 38(3):1012–1032, 2008.
- [161] Svein Høgemo, Jan Arne Telle, and Erlend Raa Vågset. Linear MIM-width of trees. In Ignasi Sau and Dimitrios M. Thilikos, editors, *Proceedings of the 45th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2019)*, volume 11789 of *LNCS*, pages 218–231. Springer, 2019.
- [162] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k -SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.
- [163] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.
- [164] Robert W. Irving and David F. Manlove. The b-chromatic number of a graph. *Discrete Applied Mathematics*, 91(1-3):127–141, 1999.
- [165] Tetsuya Ishizeki, Yota Otachi, and Koichi Yamazaki. An improved algorithm for the longest induced path problem on k-chordal graphs. *Discrete Applied Mathematics*, 156(15):3057–3059, 2008.

- [166] Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.
- [167] Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. In Dimitris Fotakis, Aris Pagourtzis, and Vangelis Th. Paschos, editors, *Proceedings of the 10th International Conference on Algorithms and Complexity (CIAC 2017)*, volume 10236 of *LNCS*, pages 345–356. Springer, 2017.
- [168] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width I. Induced path problems. *Discrete Applied Mathematics*, 278:153–168, 2020.
- [169] Lars Jaffke and Paloma T. Lima. A complexity dichotomy for critical values of the b -chromatic number of graphs. *Theoretical Computer Science*, 815:182–196, 2020.
- [170] Lars Jaffke, Paloma T. Lima, and Daniel Lokshtanov. b -Coloring parameterized by clique-width. *CoRR*, abs/2003.04254, 2020.
- [171] Robert E. Jamison and Henry Martyn Mulder. Constant tolerance intersection graphs of subtrees of a tree. *Discrete Mathematics*, 290(1):27–46, 2005.
- [172] Bart M. P. Jansen, Venkatesh Raman, and Martin Vatshelle. Parameter ecology for feedback vertex set. *Tsinghua Sci. Technol.*, 19(4):387–409, 2014.
- [173] Klaus Jansen. Complexity results for the optimum cost chromatic partition problem, 1997.
- [174] Vít Jelínek. The rank-width of the square grid. *Discrete Applied Mathematics*, 158(7):841–850, 2010.
- [175] Frank Kammer and Torsten Tholey. A lower bound for the treewidth of k -outerplanar graphs. Technical Report 2009-7, Institut für Informatik, Universität Augsburg, 2009.
- [176] Dong Yeap Kang, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. A width parameter useful for chordal and co-comparability graphs. *Theoretical Computer Science*, 704:1–17, 2017.
- [177] Iyad Kanj, Michael Pelsmajer, and Marcus Schaefer. Parameterized algorithms for feedback vertex set. In Rodney G. Downey, Michael R. Fellows, and Frank K. H. A. Dehne, editors, *Proceedings of the First International Workshop on Parameterized and Exact Computation (IWPEC 2004)*, volume 3162 of *LNCS*, pages 235–247. Springer, 2004.
- [178] Haim Kaplan and Yahav Nussbaum. A simpler linear-time recognition of circular-arc graphs. *Algorithmica*, 61(3):694–737, 2011.

- [179] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.
- [180] Ioannis Katsikarelis. Computing bounded-width tree and branch decompositions of k -outerplanar graphs. *CoRR*, abs/1301.5896, 2013.
- [181] Ken-ichi Kawarabayashi and Yusuke Kobayashi. A linear time algorithm for the induced disjoint paths problem in planar graphs. *Journal of Computer and System Sciences*, 78(2):670–680, 2012.
- [182] Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424–435, 2012.
- [183] J. Mark Keil. Finding hamiltonian circuits in interval graphs. *Information Processing Letters*, 20(4):201–206, 1985.
- [184] Christoph W. Kessler. Scheduling expression DAGs for minimal register need. *Computer Languages*, 24(1):33–53, 1998.
- [185] Sulamita Klein and Aurora Morgana. On clique-colouring of graphs with few P_4 's. *Journal of the Brazilian Computer Society*, 18(2):113–119, 2012.
- [186] Jon M. Kleinberg and Éva Tardos. *Algorithm Design*. Addison-Wesley, 2006.
- [187] Ton Kloks. *Treewidth: Computations and approximations*, volume 842 of *LNCS*. Springer, 1994.
- [188] Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Information Processing Letters*, 74(3-4):115–121, 2000.
- [189] Donald E. Knuth. Big omicron and big omega and big theta. *ACM SIGACT News*, 8(2):18–24, 1976.
- [190] Donald E. Knuth. *The Art of Computer Programming, Volume 4 Fascicle 6: Satisfiability*. Addison-Wesley, 2015.
- [191] Daniel Kobler and Udi Rotics. Polynomial algorithms for partitioning problems on graphs with fixed clique-width (extended abstract). In S. Rao Kosaraju, editor, *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2001)*, pages 468–476. ACM/SIAM, 2001.
- [192] Daniel Kobler and Udi Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Applied Mathematics*, 126(2-3):197–221, 2003.
- [193] Nicholas Korpelainen, Vadim V. Lozin, and Colin Mayhill. Split permutation graphs. *Graphs and Combinatorics*, 30(3):633–646, 2014.

- [194] Jan Kratochvíl, Paul D. Manuel, and Mirka Miller. Generalized domination in chordal graphs. *Nordic Journal of Computing*, 2(1):41–50, 1995.
- [195] Jan Kratochvíl and Zsolt Tuza. On the complexity of bicoloring clique hypergraphs of graphs. *Journal of Algorithms*, 45(1):40–54, 2002.
- [196] Jan Kratochvíl, Zsolt Tuza, and Margit Voigt. On the b-chromatic number of graphs. In Ludek Kucera, editor, *Proceedings of the 28th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2002)*, volume 2573 of *LNCS*, pages 310–320. Springer, 2002.
- [197] Dieter Kratsch, Haiko Müller, and Ioan Todinca. Feedback vertex set and longest induced path on AT-free graphs. In Hans L. Bodlaender, editor, *Proceedings of the 29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2003)*, volume 2880 of *LNCS*, pages 309–321, 2003.
- [198] Manuel Lafond. On strongly chordal graphs that are not leaf powers. In Hans L. Bodlaender and Gerhard J. Woeginger, editors, *Proceedings of the 43rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2017)*, volume 10520 of *LNCS*, pages 386–398. Springer, 2017.
- [199] Jens Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *Journal of Algorithms*, 20(1):20–44, 1996.
- [200] Jens Lagergren and Stefan Arnborg. Finding minimal forbidden minors using a finite congruence. In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo, editors, *Proceedings of the 18th International Colloquium on Automata, Languages and Programming (ICALP 1991)*, volume 510 of *LNCS*, pages 532–543. Springer, 1991.
- [201] Michael Lampis. Model checking lower bounds for simple graphs. *Logical Methods in Computer Science*, 10(1), 2014.
- [202] Michael Lampis. Finer tight bounds for coloring on clique-width. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *LIPICS*, pages 86:1–86:14. Schloss Dagstuhl, 2018.
- [203] Renu Laskar and Jeremy Lyle. Fall colouring of bipartite graphs and cartesian products of graphs. *Discrete Applied Mathematics*, 157(2):330–338, 2009.
- [204] Juho Lauri and Christodoulos Mitillos. Complexity of fall coloring for restricted graph classes. In Charles J. Colbourn, Roberto Grossi, and Nadia Pisanti, editors, *Proceedings of the 30th International Workshop on Combinatorial Algorithms (IWOCA 2019)*, volume 11638 of *LNCS*, pages 352–364. Springer, 2019.

- [205] Cornelis Gerrit Lekkerkerker and Johan Christoph Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962.
- [206] Zuosong Liang, Erfang Shan, and Liying Kang. The clique-perfectness and clique-coloring of outer-planar graphs. *Journal of Combinatorial Optimization*, 38(3):794–807, 2019.
- [207] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.
- [208] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Transactions on Algorithms*, 14(2):1–30, 2018.
- [209] László Lovász. *Combinatorial Problems and Exercises*. North-Holland Publishing Co., 1993.
- [210] Jeremy Lyle, Nate Drake, and Renu Laskar. Independent domatic partitioning or fall coloring of strongly chordal graphs. *Congressus Numerantium*, 172:149–159, 2005.
- [211] Tze-Heng Ma and Jeremy P. Spinrad. On the 2-chain subgraph cover and related problems. *Journal of Algorithms*, 17(2):251–268, 1994.
- [212] Lucas Pierzan Magalhães. Tópicos em b-continuidade: Operações em grafos e grafos distância-hereditários. Master’s thesis, Universidade Federal do Rio de Janeiro, Brasil, 2012.
- [213] Nadimpalli V. R. Mahadev and Uri N. Peled. *Threshold Graphs and Related Topics*. Elsevier, 1995.
- [214] Johann A. Makowsky and Udi Rotics. On the clique-width of graphs with few P_4 ’s. *International Journal of Foundations of Computer Science*, 10(03):329–348, 1999.
- [215] Federico Mancini. *Graph Modification Problems Related to Graph Classes*. PhD thesis, University of Bergen, Norway, 2008.
- [216] Dániel Marx. Complexity of clique coloring and related problems. *Theoretical Computer Science*, 412(29):3487–3500, 2011.
- [217] David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983.
- [218] Ross M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147, 2003.

- [219] Ross M. McConnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241, 1999.
- [220] Stefan Mengel. Lower bounds on the mim-width of some graph classes. *Discrete Applied Mathematics*, 248:28–32, 2018.
- [221] George B. Mertzios and Derek G. Corneil. Vertex splitting and the recognition of trapezoid graphs. *Discrete Applied Mathematics*, 159(11):1131–1147, 2011.
- [222] George B. Mertzios, Ignasi Sau, and Shmuel Zaks. The recognition of tolerance and bounded tolerance graphs. *SIAM Journal on Computing*, 40(5):1234–1257, 2011.
- [223] Christodoulos Mitillos. *Topics in Graph Fall-Coloring*. PhD thesis, Illinois Institute of Technology, 2016.
- [224] Bojan Mohar and Riste Skrekovski. The Grötzsch theorem for the hypergraph of maximal cliques. *Electronic Journal of Combinatorics*, 6(1):128, 1999.
- [225] Hannes Moser and Somnath Sikdar. The parameterized complexity of the induced matching problem. *Discrete Applied Mathematics*, 157(4):715–727, 2009.
- [226] Haiko Müller. Hamiltonian circuits in chordal bipartite graphs. *Discrete Mathematics*, 156(1-3):291–298, 1996.
- [227] John H. Muller and Jeremy Spinrad. Incremental modular decomposition. *Journal of the ACM*, 36(1):1–19, 1989.
- [228] Assaf Natanzon, Ron Shamir, and Roded Sharan. Complexity classification of some edge modification problems. *Discrete Applied Mathematics*, 113(1):109–128, 2001.
- [229] Sridhar Natarajan and Alan P. Sprague. Disjoint paths in circular arc graphs. *Nordic Journal of Computing*, 3(3):256–270, 1996.
- [230] Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.
- [231] Nestor V. Nestoridis and Dimitrios M. Thilikos. Square roots of minor closed graph classes. *Discrete Applied Mathematics*, 168:34–39, 2014.
- [232] Ragnar Nevries and Christian Rosenke. Towards a characterization of leaf powers by clique arrangements. *Graphs and Combinatorics*, 32(5):2053–2077, 2016.
- [233] Stavros D. Nikolopoulos and Leonidas Palios. Detecting holes and antiholes in graphs. *Algorithmica*, 47(2):119–138, 2007.

- [234] Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. On graph powers for leaf-labeled trees. *Journal of Algorithms*, 42(1):69–108, 2002.
- [235] Sang-il Oum. Rank-width and vertex-minors. *Journal of Combinatorial Theory, Series B*, 95(1):79–100, 2005.
- [236] Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Transactions on Algorithms*, 5(1):1–20, 2008.
- [237] Sang-il Oum. Computing rank-width exactly. *Information Processing Letters*, 109(13):745–748, 2009.
- [238] Sang-il Oum. Rank-width: Algorithmic and structural results. *Discrete Applied Mathematics*, 231:15–24, 2017.
- [239] Sang-il Oum, Sigve Hortemo Sæther, and Martin Vatshelle. Faster algorithms for vertex partitioning problems parameterized by clique-width. *Theoretical Computer Science*, 535:16–24, 2014.
- [240] Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.
- [241] B. S. Panda and D. Pradhan. NP-completeness of Hamiltonian cycle problem on rooted directed path graphs. *CoRR*, abs/0809.2443, 2008.
- [242] Fahad Panolan, Geevarghese Philip, and Saket Saurabh. On the parameterized complexity of b-chromatic number. *Journal of Computer and System Sciences*, 84:120–131, 2017.
- [243] Irena Penev. Perfect graphs with no balanced skew-partition are 2-clique-colorable. *Journal of Graph Theory*, 81(3):213–235, 2016.
- [244] Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(4):757–771, 2003.
- [245] Michał Pilipczuk. Computing tree decompositions. In *Treewidth, Kernels, and Algorithms. Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *LNCS*, pages 189–213. Springer, 2020.
- [246] Prasad Raghavendra and David Steurer. Graph expansion and the unique games conjecture. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 755–764. ACM, 2010.

- [247] Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for undirected feedback vertex set. In Prosenjit Bose and Pat Morin, editors, *Proceedings of the 13th International Symposium on Algorithms and Computation (ISAAC 2002)*, volume 2518 of *LNCS*, pages 241–248. Springer, 2002.
- [248] Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Transactions on Algorithms*, 2(3):403–415, 2006.
- [249] Michaël Rao. *Décompositions de graphes et algorithmes efficaces*. PhD thesis, University of Metz, 2006.
- [250] Michaël Rao. Clique-width of graphs defined by one-vertex extensions. *Discrete Mathematics*, 308(24):6157–6165, 2008.
- [251] Bruce A. Reed. Finding approximate separators and computing tree width quickly. In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC 1992)*, pages 221–228. ACM, 1992.
- [252] H. N. de Ridder. Information system on graph classes and their inclusions. graphclasses.org, accessed May 2020.
- [253] Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.
- [254] Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.
- [255] Neil Robertson and Paul D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991.
- [256] Neil Robertson and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.
- [257] Hein Röhrlig. Tree decomposition: A feasibility study. Master’s thesis, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1998.
- [258] Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.
- [259] Irena Rusu and Jeremy Spinrad. Forbidden subgraph decomposition. *Discrete mathematics*, 247(1-3):159–168, 2002.

- [260] Sigve Hortemo Sæther and Martin Vatshelle. Hardness of computing width parameters based on branch decompositions over the vertex set. *Theoretical Computer Science*, 615:120–125, 2016.
- [261] Leonardo Sampaio. *Algorithmic aspects of graph colourings heuristics*. PhD thesis, Université Nice Sophia Antipolis, 2012.
- [262] Ravi Sethi. Complete register allocation problems. *SIAM Journal on Computing*, 4(3):226–248, 1975.
- [263] Ravi Sethi and Jeffrey D. Ullman. The generation of optimal code for arithmetic expressions. *Journal of the ACM*, 17(4):715–728, 1970.
- [264] Paul Seymour. How the proof of the strong perfect graph conjecture was found. *Gazette des Mathématiciens*, 109:69–83, 2006.
- [265] Paul Seymour. The origin of the notion of treewidth. Theoretical Computer Science Stack Exchange, 2014. [Question 27317](#) (version: 2014-11-03).
- [266] Erfang Shan, Zuosong Liang, and Liying Kang. Clique-transversal sets and clique-coloring in planar graphs. *European Journal of Combinatorics*, 36:367–376, 2014.
- [267] Sadia Sharmin. *Practical Aspects of the Graph Parameter Boolean-width*. PhD thesis, University of Bergen, Norway, 2014.
- [268] Ana Silva. Graphs with small fall-spectrum. *Discrete Applied Mathematics*, 254:183–188, 2019.
- [269] Ana Shirley Ferreira da Silva. *The b-chromatic number of some tree-like graphs*. PhD thesis, Université Joseph-Fourier - Grenoble I, 2010.
- [270] Jeremy P. Spinrad. Recognition of circle graphs. *Journal of Algorithms*, 16(2):264–282, 1994.
- [271] R. Sritharan. A linear time algorithm to recognize circular permutation graphs. *Networks*, 27(3):171–174, 1996.
- [272] Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM Journal on Discrete Mathematics*, 10(4):529–550, 1997.
- [273] Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Cutwidth I: A linear time fixed parameter algorithm. *Journal of Algorithms*, 56(1):1–24, 2005.

- [274] Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Cutwidth II: Algorithms for partial w -trees of bounded degree. *Journal of Algorithms*, 56(1):25–49, 2005.
- [275] Wolfgang Thomas. Languages, automata, and logic. In *Handbook of formal languages*, pages 389–455. Springer, 1997.
- [276] Ioan Todinca. Coloring powers of graphs of bounded clique-width. In Hans L. Bodlaender, editor, *Proceedings of the 29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2003)*, volume 2880 of *LNCS*, pages 370–382. Springer, 2003.
- [277] Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series-parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1982.
- [278] Jean-Marie Vanherpe. Clique-width of partner-limited graphs. *Discrete Mathematics*, 276(1-3):363–374, 2004.
- [279] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC 2012)*, pages 887–898. ACM, 2012.
- [280] Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In Thore Husfeldt and Iyad A. Kanj, editors, *Proceedings of the 10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*, volume 43 of *LIPICS*, pages 17–29. Schloss Dagstuhl, 2015.
- [281] Martin Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, 2012.
- [282] Martin Vatshelle. personal communication, 2020.
- [283] Clara Inés Betancur Velasquez, Flavia Bonomo, and Ivo Koch. On the b-coloring of P_4 -tidy graphs. *Discrete Applied Mathematics*, 159(1):60 – 68, 2011.
- [284] Klaus Wagner. Über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114(1):570–590, 1937.
- [285] James Richard Walter. *Representations of Rigid Cycle Graphs*. PhD thesis, Wayne State University, Detroit, USA, 1972.
- [286] Egon Wanke. k -NLC graphs and polynomial algorithms. *Discrete Applied Mathematics*, 54:251–266, 1994.

- [287] Yu Wu, Per Austrin, Toniann Pitassi, and David Liu. Inapproximability of treewidth and related problems. *Journal of Artificial Intelligence Research*, 49:569–600, 2014.
- [288] Mihalis Yannakakis. The complexity of the partial order dimension problem. *SIAM Journal on Algebraic Discrete Methods*, 3(3):351–358, 1982.

Part V

Appendix

A

Dictionary

Logical Expressions. A *logical expression* is a string of symbols that evaluates either to **true** or to **false**. For two logical expressions ϕ and ψ , we denote by ' $\phi \wedge \psi$ ' the logical expression that evaluates to **true** if and only if ϕ and ψ both evaluate to **true**, and by ' $\phi \vee \psi$ ' the logical expression that evaluates to **true** if and only if ϕ or ψ (meaning at least one of ϕ and ψ) evaluates to **true**. We let ' $\neg\phi$ ' be the *negation* of ϕ , i.e. the logical expression that evaluates to **true** if and only if ϕ evaluates to **false**. We let ' $\phi \Rightarrow \psi$ ' denote that ϕ *implies* ψ , i.e. it holds that whenever ϕ evaluates to **true**, then ψ evaluates to **true**. Formally, $\phi \Rightarrow \psi := \neg(\phi \wedge \neg\psi)$. We note that $\phi \Rightarrow \psi$ may be written as $\psi \Leftarrow \phi$, and we use the shorthand ' $\phi \Leftrightarrow \psi$ ' for ' $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$ ', in which case we call ϕ and ψ *equivalent*.

Sets/Subsets. A well-defined collection of distinct objects $\Omega = \{\omega_1, \dots, \omega_n\}$ is called a *set*, and $\omega_1, \dots, \omega_n$ are called its *elements*. If n is finite, then Ω is called finite, and we denote by $|\Omega|$ the *size/cardinality* of Ω , i.e. $|\Omega| := n$. For an element ω and a set Ω , we write ' $\omega \in \Omega$ ' if and only if ω is an element of Ω . We use the shorthand ' $\omega \notin \Omega$ ' for ' $\neg(\omega \in \Omega)$ '. A set Γ is called a *subset* of Ω if for all $\omega \in \Gamma$, we have that $\omega \in \Omega$, in which case we write ' $\Gamma \subseteq \Omega$ '. If $\Gamma \subseteq \Omega$ and $\Omega \subseteq \Gamma$ then we say that Γ and Ω are *equal*, and we write ' $\Omega = \Gamma$ '. If Γ and Ω are not equal, then we write ' $\Omega \neq \Gamma$ '. If $\Gamma \subseteq \Omega$ and $\Gamma \neq \Omega$, we call Γ a *strict subset* of Ω and write ' $\Gamma \subset \Omega$ '. For two sets Ω and Γ , we let

- $\Omega \cup \Gamma := \{\omega \mid \omega \in \Omega \vee \omega \in \Gamma\}$ be the *union of Ω and Γ* ,
- $\Omega \cap \Gamma := \{\omega \mid \omega \in \Omega \wedge \omega \in \Gamma\}$ be the *intersection of Ω and Γ* ,
- $\Omega \setminus \Gamma := \{\omega \mid \omega \in \Omega \wedge \omega \notin \Gamma\}$ be the *difference of Ω from Γ* , and
- $\Omega \Delta \Gamma := (\Omega \setminus \Gamma) \cup (\Gamma \setminus \Omega)$ be the *symmetric difference of Ω and Γ* .

For a set Ω , we denote by 2^Ω the set of all subsets of Ω , and given an integer $k \leq |\Omega|$, we denote by $\binom{\Omega}{k}$ the set of all size- k subsets of Ω .

Numbers. We denote the set of natural numbers by $\mathbb{N} := \{0, 1, 2, \dots\}$, and the positive natural numbers by $\mathbb{N}^+ := \mathbb{N} \setminus \{0\}$. For $m, n \in \mathbb{N}$ with $m \leq n$, we let $[m..n] := \{m, m+1, \dots, n\}$, and for $n \in \mathbb{N}^+$, we let $[n] := [1..n]$. We denote by \mathbb{R} the set of real numbers, and we let by \mathbb{R}^+ the positive real numbers and by \mathbb{R}_0^+ the non-negative real numbers.

Restricted Subsets. For a set Ω and a property ψ that an element in Ω may have, we denote by Ω_ψ the biggest subset of Ω where ψ is satisfied for all elements. For instance, $\mathbb{N}_{\leq k}^+$ denotes the set $\{1, 2, \dots, k\}$.

Maps/Functions. Let X and Y be sets. An object $\phi: X \rightarrow Y$ which assigns each $x \in X$ some $\phi(x) \in Y$ is called a *map/function from X to Y* . For a subset $X' \subseteq X$, we let $\phi(X') := \{\phi(x') \mid x' \in X'\}$. Two maps $\phi: X \rightarrow Y$ and $\psi: X \rightarrow Y$ are called *equal*, in symbols ' $\phi = \psi$ ' if for all $x \in X$, $\phi(x) = \psi(x)$.

Let X, Y , and Z be sets and $\phi: X \rightarrow Y$ and $\psi: Y \rightarrow Z$ be maps. The *composition* of ϕ and ψ , denoted by $\phi \circ \psi$, maps each $x \in X$ to $\psi(\phi(x))$, i.e. we first apply ϕ and then ψ . For a set X , we let $\text{id}_X: X \rightarrow X$ be the *identity map* on X which maps each element of X to itself, i.e. for all $x \in X$, $\text{id}_X(x) = x$.

We call a map $\phi: X \rightarrow Y$ *injective (an injection)* if $\phi(x) = \phi(y) \Rightarrow x = y$ and *surjective (a surjection)* if for each $y \in Y$, there is some $x \in X$ such that $\phi(x) = y$. If a map ϕ is both injective and surjective, then it is *bijective (a bijection)*. If ϕ is bijective, then its *inverse (inverse map)* $\phi^{-1}: Y \rightarrow X$ which is such that $\phi \circ \phi^{-1} = \text{id}_X$ is well-defined.

With a slight abuse of notation, given a non-bijective map $\phi: X \rightarrow Y$ and an element $y \in Y$, we also denote by $\phi^{-1}(y)$ the set $X' \subseteq X$ such that for all $x' \in X'$, $\phi(x') = y$.

For a map $\phi: X \rightarrow Y$ and a set $X' \subseteq X$, we denote by $\phi|_{X'}$ the *restriction* of ϕ to X' , i.e. the map $\phi|_{X'}: X' \rightarrow Y$ which is such that for all $x' \in X'$, $\phi|_{X'}(x') = \phi(x')$.

Equivalence Relations. Let Ω be a set. A *binary relation* R on Ω is a set of ordered pairs $R \subseteq \Omega \times \Omega$. A binary relation R is an *equivalence relation* if it is *reflexive* ($\forall x \in \Omega: (x, x) \in R$), *symmetric* ($\forall x, y \in \Omega: (x, y) \in R \Leftrightarrow (y, x) \in R$) and *transitive* ($\forall x, y, z \in \Omega: (x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$). For an element $x \in \Omega$ the *equivalence class of x* , denoted by $[x]_\sim$, is the set $[x]_\sim := \{y \in \Omega \mid (x, y) \in R\}$. We denote the set of all equivalence classes of R by Ω/R . Given an equivalence relation R , we usually use the notation $x \sim y$ whenever $(x, y) \in R$, and denote the set of equivalence classes of \sim by Ω/\sim .

Partial Orders. Let X be a set. A *partial order* of X is a binary relation R on X . We say that a partial order R is *strict* if there is no pair $x, y \in X$ such that $(x, y) \in R$ and $(y, x) \in R$.

Linear Orders. Let X be a set of size n . A *linear order* of X is a bijection $\pi: X \rightarrow [n]$. Given a subset $X' \subseteq X$ of size $n' \leq n$, we define the *restriction of π to X'* as the bijection $\pi|_{X'}: X' \rightarrow [n']$ which is such that for all $x', y' \in X'$, $\pi|_{X'}(x') < \pi|_{X'}(y')$ if and only if $\pi(x') < \pi(y')$.

A.1 Graphs

An *undirected graph* or simply *graph* G is a pair of a vertex set $V(G)$ and an edge set $E(G)$. For our purposes, we can consider $E(G)$ a subset of pairs of $V(G)$, unless stated otherwise. For more formal and general definitions, we refer to [45]. We use the notation $|G| := |V(G)|$ and $\|G\| := |E(G)|$. For an edge $\{u, v\} \in E(G)$, we use the shorthand notation ‘ uv ’. For a set of edges $F \subseteq E(G)$, we denote by $V(F)$ the set of vertices that are contained in the edges of F , i.e. $V(F) := \bigcup_{uv \in F} \{u, v\}$. A *cut* of a graph is a 2-partition of its vertex set.

(Induced) Subgraphs. For graphs G and H we say that H is a *subgraph* of G , denoted by $H \subseteq G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For a vertex set $X \subseteq V(G)$, we denote by $G[X]$ the subgraph of G *induced* by X , i.e. the graph on vertex set X and edge set $\{xy \in E(G) \mid x, y \in X\}$. We use the notation $G - X := G[V(G) \setminus X]$ and for a set of edges $F \subseteq E(G)$, $G - F := (V(G), E(G) \setminus F)$. For a vertex $v \in V(G)$ (an edge $e \in E(G)$), we use the shorthand ‘ $G - v$ ’ for ‘ $G - \{v\}$ ’ (‘ $G - e$ ’ for ‘ $G - \{e\}$ ’).

Neighborhoods/Degree. Let G be a graph. For a vertex $v \in V(G)$, we denote by $N_G(v)$ the *open neighborhood* of v , i.e. $N_G(v) := \{w \mid vw \in E(G)\}$, and by $N_G[v]$ the *closed neighborhood* of v , i.e. $N_G[v] := \{v\} \cup N_G(v)$. The *degree* of v is the size of its open neighborhood, i.e. $\deg_G(v) := |N_G(v)|$. For a set of vertices $X \subseteq V(G)$, we let $N_G(X) := \bigcup_{v \in X} N_G(v) \setminus X$ and $N_G[X] := N_G(X) \cup X$. We use the shorthand notations ‘ N ’ and ‘ \deg ’ for ‘ N_G ’ and ‘ \deg_G ’, respectively, if G is clear from the context. For a graph G , we denote by $\Delta(G) := \max_{v \in V(G)} \deg(v)$ the *maximum degree* of G . We say that G is *subcubic* if $\Delta(G) \leq 3$.

Twins. Let G be a graph and let $u, v \in V(G)$ be two vertices. We say that u and v are *false twins* if $N_G(u) = N_G(v)$, and that u and v are *true twins* if $N_G[u] = N_G[v]$.

Connectivity/Acyclicity/Distance. A graph G is called *connected* if for each 2-partition (X, Y) of $V(G)$ with $X \neq \emptyset$ and $Y \neq \emptyset$, there is an edge $xy \in E(G)$ such that $x \in X$ and $y \in Y$. The inclusion-wise maximal connected subgraphs of a graph G are called its *connected components*, and denoted by $\text{cc}(G)$. A connected graph all of whose vertices have degree two is called a *cycle*. A graph that does not contain a cycle as a subgraph is called a *forest* (or *acyclic*) and

a connected forest is a *tree*. The vertices of degree one in a tree are called the *leaves* and the remaining vertices the *internal* vertices. A tree of maximum degree two is called a *path* and the leaves of a path are called *endpoints*. The *length* of a path P is $|E(P)|$, i.e. the number of its edges. A tree T is called a *caterpillar* if it contains a subgraph $P \subseteq T$ such that P is a path and each vertex in $V(T) \setminus V(P)$ has a neighbor in $V(P)$. A caterpillar whose path has length 0 is a *star*. Given a graph G and two of its vertices $u, v \in V(G)$, the *distance* from u to v in G , denoted by $\text{dist}_G(u, v)$, is the smallest length of any path $P \subseteq G$ with endpoints u and v .

Rooted Trees. A *rooted tree* is a tree T with one designated vertex $r \in V(T)$, called the *root* of T , which induces an ancestral relation on the vertices of T . For each non-root node $v \in V(T) \setminus \{r\}$, let $p \in V(T)$ be the vertex adjacent to v in the (unique) path from r to v , then we call p the *parent* of v and we call $N_T(v) \setminus \{p\}$ the *children* of v . The *subtree of T rooted at v* , denoted by T_v , is the connected component of $T - pv$ containing v . The neighbors of r are called its *children*, and we define $T_r := T$. For each node $v \in V(T)$, its *height* in T is the length of any path from v to a leaf of T_v . A rooted tree is called a *full binary tree* if each of its vertices has either two or zero children.

Separators. For a graph G , a *separator* of G is a vertex set $S \subseteq V(G)$ such that $G - S$ has more connected components than G . A *cut vertex* is a vertex $v \in V(G)$ such that $\{v\}$ is a separator of G .

Matchings. A set of edges $M \subseteq E(G)$ is called a *matching*, if no pair of edges in M shares an endpoint. A matching is called *induced* if $G[V(M)] = (V(M), M)$, i.e. if there are no other edges than the edges in M in the subgraph of G induced by $V(M)$.

Cliques/Independence/Vertex Cover/Domination. A graph G is called *complete*, if $E(G) = \binom{V(G)}{2}$. A set of vertices $C \subseteq V(G)$ is called a *clique* if $G[C]$ is a complete graph. A graph G is called *empty*, if $E(G) = \emptyset$. A set of vertices $I \subseteq V(G)$ is called an *independent set* if $G[I]$ is empty. A set of vertices $S \subseteq V(G)$ is called a *vertex cover* if $G - S$ is empty. A set of vertices $D \subseteq V(G)$ is called a *dominating set* if $N_G[D] = V(G)$.

Maximum/Minimum Π Sets. For a property Π of vertex sets of a graph G , we say that a set $X \subseteq V(G)$ is a *maximum Π* (*minimum Π*) set, if for any other vertex set $Y \subseteq V(G)$ with property Π , $|X| \geq |Y|$ ($|X| \leq |Y|$). For instance, if Π is the property of being an independent set, then given a graph G and a vertex subset $X \subseteq V(G)$, we say that X is a *maximum independent set* if for each independent set $Y \subseteq V(G)$ in G , $|X| \geq |Y|$.

Maximal/Minimal Π Sets. For a property Π of vertex sets of a graph G , we say that a set $X \subseteq V(G)$ is a *maximal Π* (*minimal Π*) set, if there is no strict

superset $Y \supset X$ (subset $Y \subset X$) with property Π .

Bipartite Graphs. A graph G is called *bipartite* if there is a 2-partition (X, Y) of $V(G)$ such that X and Y are independent sets, and we call a bipartite graph G with partition (X, Y) *complete bipartite* if $E(G) = \{xy \mid x \in X, y \in Y\}$. For two disjoint vertex sets $X, Y \subseteq V(G)$, we denote by $G[X, Y]$ the *bipartite subgraph of G induced by (X, Y)* , i.e. $G[X, Y] := (X \cup Y, E(G) \cap \{xy \mid x \in X, y \in Y\})$.

Isomorphisms. Let G and H be two graphs. A bijection $\varphi: V(G) \rightarrow V(H)$ is called an *isomorphism from G to H* if for all $u, v \in E(G)$, $uv \in E(G)$ if and only if $\varphi(u)\varphi(v) \in E(H)$. If there is an isomorphism from G to H then say that G and H are *isomorphic*.

(Vertex) Colorings. Let G be a graph. A *vertex coloring* of G (with k colors) is a partition $\mathcal{C} = (C_1, \dots, C_k)$ of G . The sets C_1, \dots, C_k are called the *color classes*. A *proper coloring* is a coloring in which each color class is an independent set. A *b-coloring* (with k colors) of a graph G is a pair (\mathcal{C}, B) of a proper coloring $\mathcal{C} = (C_1, \dots, C_k)$ and a set of vertices $B \subseteq V(G)$, called *b-vertices*, such that for each $i \in [k]$, B contains precisely one vertex from C_i , say v_i , such that for all $j \neq i$, $N_G(v_i) \cap C_j \neq \emptyset$. A *clique coloring* is a coloring \mathcal{C} containing no *monochromatic maximal clique*, i.e. there is no maximal clique X in G such that for some $C \in \mathcal{C}$, $X \subseteq C$. We may alternatively represent a vertex coloring $\mathcal{C} = (C_1, \dots, C_k)$ as a map $\gamma: V(G) \rightarrow [k]$ such that for all $v \in V(G)$, $v \in C_{\gamma(v)}$.

Multigraphs. A *multigraph* is a pair of a set of vertices $V(G)$ and a *multiset* of size-2 subsets of $V(G)$, called the edges of G and denoted by $E(G)$.

Subdivisions. Let G be a (multi-) graph. An *edge subdivision* of an edge $e \in E(G)$ with endpoints u and v is the operation of adding to G a new vertex x and replacing the edge e by a path with endpoints u and v , consisting of an edge between u and x and an edge between x and v . We call the vertex x a *subdivision vertex*. A graph G' is called a *subdivision of G* if it can be obtained from G by a series of edge subdivisions.

Contractions. Let G be a (multi-) graph and $S \subseteq V(G)$ be a subset of its vertices. The *contraction* of S in G is the operation of adding a vertex x_S to G , making it adjacent to $N_G(S)$, and then removing S from G . An *edge contraction* is the operation of contracting the endpoints of an edge.

Minors/Topological Minors. A graph H is called *minor* of a graph G if we can obtain a graph isomorphic to H by performing a series of vertex deletions, edge deletions, or edge contractions on G . The *dissolution* of a vertex v of degree two in G is the operation of making the two neighbors of v adjacent and removing v . (Note that this operation can be viewed as ‘undoing a subdivision’.) A

graph H is called a *topological minor* of a graph G if we can obtain a graph isomorphic to H by performing a series of vertex deletions, edge deletions, and vertex dissolutions on G .

Spanning Subgraphs/Hamiltonian Paths and Cycles. Let G be a graph and H be a subgraph of G . We say that H is a *spanning* subgraph if $V(H) = V(G)$. A *Hamiltonian path* is a spanning subgraph that is a path and a *Hamiltonian cycle* is a spanning subgraph that is a cycle. A *spanning tree* is a spanning subgraph that is a tree.

Notation for Special Graphs. Let $n \in \mathbb{N}$. We may denote by ' K_n ' a complete graph on n vertices, by ' C_n ' a cycle on n vertices, and by ' P_n ' a path on n vertices. Let additionally $m \in \mathbb{N}$. We may denote by ' $K_{m,n}$ ' a complete bipartite graph whose 2-partition has parts of size m and n .

A.2 Digraphs

A *directed graph* (or *digraph*) G is a pair of a set of vertices $V(G)$ and a set of ordered pairs of vertices, called *arcs*, $A(G) \subseteq V(G) \times V(G)$. We say that an arc $a = (u, v) \in A(G)$ is directed from u to v , and we call u the *tail* of a and v the *head* of a . We use the shorthand ' uv ' for ' (u, v) '. We use the notations from the paragraph '(Induced) Subgraphs' about undirected graphs (page 291) in the context of digraphs as well.

Walks/Paths/DAGs. A sequence of vertices v_1, \dots, v_r is called a *walk* in G if for all $i \in [r - 1]$, $v_i v_{i+1} \in A(G)$. A *path* is a walk whose vertices are pairwise distinct. A *cycle* is a walk v_1, \dots, v_r with $v_1 = v_r$ and all vertices v_1, \dots, v_{r-1} pairwise distinct. If G does not contain any cycles as a subgraph, then we call G *acyclic* or a *directed acyclic graph*, DAG for short.

Topological Orders. Let G be a DAG on n vertices. A *topological order* of G is a linear order $\pi: V(G) \rightarrow [n]$ such that for all arcs $uv \in A(G)$, we have that $\pi(u) < \pi(v)$. We denote the set of all topological orders of G by $\Pi(G)$.

Multidigraph. A *multidigraph* is a pair of a set of vertices $V(G)$ and a *multiset* of ordered pairs of $V(G)$, called the arcs of G and denoted by $A(G)$.

A.3 Asymptotics, Runtime, and Complexity

We give a brief introduction into the basic notions of asymptotics and runtime analysis of algorithms and refer to for instance [79, 186] for a more thorough introduction.

We first list the Bachmann-Landau notations \mathcal{O} , Ω , Θ , o , and ω as they are used in computer science [189]. Let $f: \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$ and $g: \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$ be two functions. (We denote by \mathbb{R}_0^+ the set of non-negative real numbers.)

- We write $f(n) = \mathcal{O}(g(n))$ if and only if *there exists* a real constant $c > 0$ and an integer n_0 such that for all $n > n_0$, $f(n) \leq c \cdot g(n)$.
- We write $f(n) = \Omega(g(n))$ if and only if *there exists* a real constant $c > 0$ and an integer n_0 such that for all $n > n_0$, $f(n) \geq c \cdot g(n)$.
- We write $f(n) = \Theta(g(n))$ if both $f(n) = \mathcal{O}(g(n))$ and $f(n) = \Omega(g(n))$.
- We write $f(n) = o(g(n))$ if and only if *for all* real constants $c > 0$ there exists an integer n_0 such that for all $n > n_0$, $f(n) \leq c \cdot g(n)$.
- We write $f(n) = \omega(g(n))$ if and only if *for all* real constants $c > 0$ there exists an integer n_0 such that for all $n > n_0$, $f(n) \geq c \cdot g(n)$.

In the traditional one-dimensional framework, the runtime of an algorithm is usually expressed as a function f of the input size, such that given an input of size n , *in the worst case*, the algorithm finishes after (at most) $f(n)$ elementary steps¹. Runtime functions are expressed *asymptotically* using the notation introduced above, and for upper bounds we usually express the function f in the form ‘ $\mathcal{O}(g(n))$ ’ meaning that $f(n) = \mathcal{O}(g(n))$.

P and NP. We denote by ‘P’ the class of problems that can be solved in polynomial time, i.e. by an algorithm running in time $\mathcal{O}(n^c)$, where n is the input size and $c \in \mathbb{N}$ some constant. We denote by ‘NP’ the class of problems whose certificates can be verified in polynomial time. The SATISFIABILITY problem which given a boolean formula over a set of variables asks whether there is an assignment of truth values to its variables so that it evaluates to `true`, is the *canonical NP-hard* problem. Any other problem Π is NP-hard if there is a polynomial-time reduction from an NP-hard problem A to Π . Here, by polynomial-time reduction from A to Π we mean a polynomial-time algorithm that given each instance of A produces an equivalent instance of Π . A problem is called *NP-complete* if it both belongs to NP and is NP-hard.

¹Note that it is not trivial to formally define what we mean by a ‘step’, so in this text we rely on some intuitive understanding. Roughly, we consider every operation that can be executed within a negligible number of CPU cycles on a machine with Random Access Memory a ‘step’.

