Fine-Grained Parameterized Algorithms on Width Parameters and Beyond

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

doctor rerum naturalium
(Dr. rer. nat.)
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
der Humboldt-Universität zu Berlin

von

M.Sc. Falko Hegerfeld

Präsidentin der Humboldt-Universität zu Berlin:
Prof. Dr. Julia von Blumenthal

Dekanin der Mathematisch-Naturwissenschaftlichen Fakultät:
Prof. Dr. Caren Tischendorf

Gutachter:
1. Prof. Dr. Stefan Kratsch
2. Prof. Dr. Sebastian Siebertz
3. Prof. Dr. Jesper Nederlof

Tag der mündlichen Prüfung: 23.08.2023

Abstract

The question at the heart of parameterized complexity is how input structure governs the complexity of a problem. We investigate this question from a fine-grained perspective and study problem-parameter-combinations with single-exponential running time, i.e., time $\alpha^k n^c$, where n is the input size, k the parameter value, and α and c are positive constants. Our goal is to determine the optimal base α for a given combination. For many connectivity problems such as CONNECTED VERTEX COVER or CONNECTED DOMINATING SET, the optimal base is known relative to treewidth. Treewidth is a fundamental graph parameter that measures the size of vertex separators necessary to fully decompose a graph and, as part of the class of width parameters, naturally admits dynamic programming algorithms.

In the first part of this thesis, we study how the optimal base changes for these connectivity problems when going to more expressive width parameters capable of capturing dense graphs by allowing for more intricate separators. We provide new parameterized dynamic programming algorithms and (conditional) lower bounds to determine the optimal base, in particular, we obtain for the parameter sequence treewidth, modular-treewidth, clique-width that the optimal base for

- CONNECTED VERTEX COVER starts at 3, increases to 5, and then to 6,
- CONNECTED DOMINATING SET starts at 4, stays at 4, and then increases to 5.

In the second part, we go beyond width parameters and study more restrictive parameterizations like depth parameters and modulators that do not allow for graphs with arbitrarily many disjoint balanced separators. For treedepth, the depth-analog of treewidth, we improve the space requirement of the algorithms for the connectivity problems from exponential to polynomial while maintaining the base by transforming the usual dynamic programming algorithms into branching algorithms. The lower bound techniques for width parameterizations do not carry over to these more restrictive parameterizations and as a result, only a few optimal bases are known. We seek to remedy this situation and move away from the connectivity problems to standard vertex-deletion problems. In particular, we show that the optimal base of ODD CYCLE TRANSVERSAL parameterized by a modulator to treewidth 2 is 3 which also implies that the optimal base relative to treedepth is 3. Additionally, we show that similar lower bounds can be obtained in the realm of dense graphs by considering modulators consisting of so-called twinclasses.

Zusammenfassung

Die Kernaufgabe der parameterisierten Komplexität ist zu verstehen, wie Eingabestruktur die Problemkomplexität beeinflusst. Wir untersuchen diese Fragestellung aus einer granularen Perspektive und betrachten Problem-Parameter-Kombinationen mit einfach exponentieller Laufzeit, d.h., Laufzeit $\alpha^k n^c$, wobei n die Eingabegröße ist, k der Parameterwert, und α und c zwei positive Konstanten sind. Unser Ziel ist es, die optimale Laufzeitbasis α für eine gegebene Kombination zu bestimmen. Für viele Zusammenhangsprobleme, wie CONNECTED VERTEX COVER oder CONNECTED DOMINATING SET, ist die optimale Basis bezüglich dem Parameter Baumweite bekannt. Die Baumweite ist ein fundamentaler Graphparameter, welcher misst, welche Größe von Knotenseparatoren notwendig ist, um einen Graphen vollständig zu zerlegen. Außerdem gehört die Baumweite zu der Klasse der Weiteparameter, welche auf natürliche Weise zu Algorithmen mit dem Prinzip der dynamischen Programmierung führen.

Im ersten Teil dieser Dissertation untersuchen wir, wie sich die optimale Laufzeitbasis für diverse Zusammenhangsprobleme verändert, wenn wir zu ausdrucksstärkeren Weiteparametern wechseln, die mit Hilfe von komplexeren Separatoren auch in der Lage sind dichte Graphen zu modellieren. Wir entwerfen neue parameterisierte Algorithmen nach dem Prinzip der dynamischen Programmierung und (bedingte) untere Schranken, um diese optimalen Basen zu bestimmen. Insbesondere zeigen wir für die Parametersequenz Baumweite, modulare Baumweite, und Cliquenweite, dass die optimale Basis von

- CONNECTED VERTEX COVER bei 3 startet, sich erst auf 5 erhöht und dann auf 6,
- CONNECTED DOMINATING SET bei 4 startet, erst bei 4 bleibt und sich dann auf 5 erhöht.

Im zweiten Teil gehen wir über Weiteparameter hinaus und analysieren restriktivere Arten von Parametern. Diese weiteren Parameter lassen sich als Tiefenparameter oder Modulatoren klassifizieren und verbieten Graphen, welche beliebig viele disjunkte balancierte Separatoren enthalten. Für die Baumtiefe, welche das Tiefenanalogon der Baumweite ist, verbessern wir den Platzbedarf der Algorithmen für Zusammenhangsprobleme von exponentiell auf polynomiell und behalten zugleich die Laufzeitbasis bei. Dies erreichen wir, indem wir die gewöhnlichen Algorithmen, welche auf dynamischer Programmierung basieren, in platzsparende Verzweigungsalgorithmen umwandeln. Die Beweistechniken für untere Schranken bezüglich Weiteparametern übertragen sich jedoch nicht zu diesen restriktiveren Parametern, deswegen sind für diesen Fall nur wenige optimale Laufzeitbasen bekannt. Wir wollen diesen Mangel beheben und wechseln von den Zusammenhangsproblemen zu gewöhnlichen Knotenlöschungsproblemen. Insbesondere zeigen wir, dass die optimale Basis von ODD CYCLE TRANSVERSAL parameterisiert mit einem Modulator zu Baumweite 2 den Wert 3 hat. Zusätzlich zeigen wir, dass sich im Bereich der dichten Graphen ähnliche untere Schranken erzielen lassen, indem man Modulatoren betrachtet, welche aus Mengen von Zwillingsknoten bestehen.

Acknowledgement

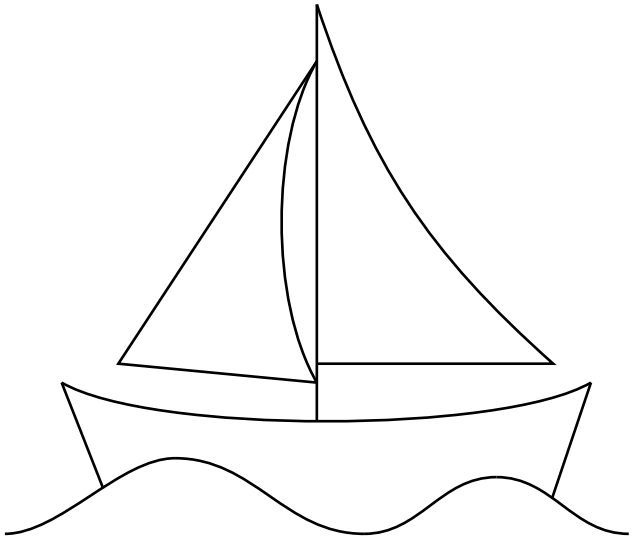
I would like to start by thanking my supervisor Stefan for his guidance and constant support. We met in Bonn, where he introduced me to the area of parameterized complexity and shortly after he moved to Berlin I followed to start my PhD under him. His deep understanding of the subject helped me to overcome a multitude of obstacles in the research. Moreover, you always had an open door and would answer any of my questions with great care. Lastly, I want to thank you for always creating a nice atmosphere in the research group and rekindling my interest in board games.

Next, I would like to thank my former colleagues Astrid, Eva-Maria, Robert, Sebastian, and in particular Florian, who accompanied me most of the way, for fun discussions at lunch and at our regular gaming evenings. I share the same appreciation for my current colleagues Katrin, Leonid, Narek, Pascal, and Vera, who all proofread parts of the thesis and provided helpful comments. I also have to thank Narek and Vera for being great coauthors; working with you two was very enjoyable. Moreover, my gratitude also goes out to Florian, Frank, Leonid, Narek, and Robert for all their teaching efforts and the many entertaining "EThI"-meetings. A further thank you to Galina who was always quick to help and never forgot to print a "Happy Birthday"-note.

Part of my research was funded by the Deutsche Forschungsgemeinschaft within the Emmy Noether-grant (KR 4286/1), which I am grateful for. Furthermore, thanks in advance to all committee members for reading my thesis.

Lastly, I want to express my gratitude to my family for their unwavering support throughout this journey and especially to Ceci, who always took great care of me. Thank you!

Dedicated to my father,
who keeps inspiring me



Contents

1 Introduction

Divide-and-conquer is one of the most fundamental algorithmic paradigms; it describes the ubiquitous principle of recursively decomposing a problem into simpler subproblems, whose solutions are then combined to a solution of the original problem. Many seminal algorithms such as fast Fourier transforms, Quicksort, Strassen's fast matrix multiplication, or the Floyd-Warshall algorithm for the ALL-PAIRS SHORTEST PATH[1] problem make use of divide-and-conquer, see the book of Cormen et al. [39]. All of these algorithms are *efficient* in the classical sense, meaning that their running time scales polynomially in the input size. We will instead deal with classically *intractable* problems, which are *not* believed to admit polynomial-time algorithms.

This work is focused on solving problems of a *graph-theoretical* nature, which are a source for many intractable problems, with divide-and-conquer algorithms. Graphs abstractly model networks arising from a *pairwise relationship*, such as social, transportation, or biological networks. We apply divide-and-conquer to graph problems in a generic and essentially problem-agnostic way by directly decomposing a given graph into smaller subgraphs and deriving the subproblems for divide-and-conquer from such a *graph decomposition*.

The obtained graph decompositions can be leveraged by an algorithm only if the interaction between subgraphs is simple, so that the resulting subproblems are easier to solve than the original problem. However, not every graph allows for such decompositions, implying that the existence of an algorithmically beneficial decomposition is imposing a *structural restriction* on the considered graphs. As we are dealing with classically intractable problems, structural restrictions are actually desired: Classical intractability only witnesses that *some* problem instances are difficult to solve; instead, we want to identify structural requirements that *simplify* the problem and, in particular, avoid these difficult instances. This is done via the language of *parameterized complexity*, where the structure of a problem instance is quantified via so-called *parameters* and a smaller parameter value indicates a greater structural constraint. The mentioned graph decompositions correspond to certain classes of parameters that we cover in more detail later.

By measuring the running time of algorithms in terms of the input size and a parameter value, we can understand how *input structure* affects *problem complexity*. Our investigation of this relationship is concentrated on problems admitting *single-exponential algorithms*, i.e., algorithms with running time $\alpha^k n^c$, where n denotes the input size, k the parameter value, and α and c are positive constants; the running time has a single-exponential dependence on the parameter value and a polynomial dependence on the input size. The constant α, also called *base* of the running time, essentially measures how complicated the interaction between the problem and the considered input structure is. To obtain a precise understanding of this interaction, we perform a *fine-grained analysis* that tries to optimize the base α as

[1] Problem names are, as is common, written in small capitals.

much as possible, ideally even proving a tight (conditional) lower bound that α cannot be improved further; in that case, we call α the *optimal base*. Our analysis usually neglects to optimize the polynomial dependence n^c, therefore we tend to write $\mathcal{O}^*(\alpha^k) = \alpha^k n^{\mathcal{O}(1)}$ for the running time instead.

When the optimality of the base α for some problem-parameter-combination is established, it is natural to consider how the base is influenced by changes to the parameter. We study various structural restrictions on the divide-and-conquer approach for graphs, corresponding to different parameters, and precisely analyze their effect on the base α. By considering more lenient structural restrictions, we obtain parameters with a greater *expressiveness*, i.e., they capture more graphs compared to a stricter parameter. This expressiveness, or *generality*, comes at the price of more complicated interactions with the input structure and thus higher algorithmic difficulty. Determining the optimal bases for a range of parameters, from restrictive to expressive, allows us to give a precise quantification on the *price of generality* in this setting. Moreover, if the optimal base remains the same for a restrictive and a more expressive parameter, we obtain a better understanding of which structural features cause the algorithmic difficulty and which structural features can be dealt with at no algorithmic cost.

We divide the considered structural restrictions on the divide-and-conquer approach into two classes. The first class consists of *width parameters*, which essentially only limit the allowed interaction between subgraphs in each step of the decomposition process. In contrast, the second class consists of *depth parameters* and *modulators*, which additionally restrict how decomposition steps are allowed to interact or even restrict the number of steps, thus possibly allowing for improved algorithms. Width and depth parameters roughly correspond to two different algorithmic instantiations of the divide-and-conquer approach; width parameters correspond to *dynamic programming* and depth parameters correspond to *branching algorithms*. In what follows, we first give a more detailed introduction to these kinds of parameters, cf. Section 1.1 and Section 1.2, and then introduce *connectivity problems*, which form the main class of graph problems studied in this thesis, cf. Section 1.3.

1.1 Width Parameters

We begin by introducing the so-called *width parameters*. It is challenging to give a formal framework that precisely[2] captures the large number of width parameters that have been defined in the literature [26, 30, 41, 54, 83, 149, 159, 160, 165, 172], so we settle for an informal explanation. Intuitively, we view the considered width parameters as arising from a recursive decomposition of a graph in the vein of divide-and-conquer.

[2] For many width parameters, one can obtain via the framework of Robertson and Seymour [161], which defines width parameters based on a symmetric set function, another width parameter that is bounded on the same graph classes. However, this can usually scale the given width parameter by at least a constant factor, e.g. we only know that clique-width is at most twice as large as module-width, see Rao [158], but even such a dependence already affects the base obtained in our fine-grained analysis of single-exponential algorithms. Eiben et al. [64] show that a natural subclass of symmetric set functions is sufficient to characterize many known width parameters in this way.

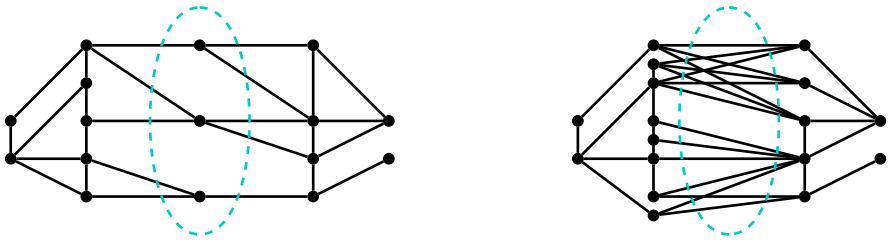


Fig. 1.1.: In the left graph, the dashed cyan ellipse highlights a vertex separator of size 3. In the right graph, the dashed cyan ellipse highlights a dense, but structured, edge separator that can be built from 3 bicliques.

Hierarchical Decomposition Scheme and Separators. To apply the divide-and-conquer paradigm to graph problems, we want to decompose the given graph into multiple pieces on which we can solve the considered problem almost independently. This is simple if the graph is not connected, then we can simply take its connected components as pieces. However, once we have done so, we need to change our decomposition strategy, since further decomposition is only possible if we allow the pieces to interact. Here, we observe that the ease of designing fast algorithms for the subproblems is correlated with how complicated the interaction between the pieces is allowed to be, so a good decomposition should limit the interaction as much as possible.

Going back to the graph, we therefore try to find a *separator*, which could consist of vertices or edges, whose removal decomposes the graph into several pieces and which sufficiently limits the interaction of the pieces. This limitation could be for one of two reasons: either the separator is small or the separator has a simple structure; the former is appropriate for *sparse graphs* and the latter for *dense, but structured, graphs*, see Figure 1.1. With the resulting pieces in hand, we simply repeat our strategy and decompose these pieces over and over, until only sufficiently easy pieces remain where the problem can be solved directly. This informal decomposition scheme naturally arranges the separators in a *hierarchical*, or tree-like, fashion, where every level of the hierarchy corresponds to another round of decomposition. Hence, we refer to such a decomposition strategy as a *hierarchical decomposition scheme*, see Figure 1.2 for an example.

From Hierarchical Decomposition Scheme to Width Parameter. To obtain a width parameter from a hierarchical decomposition scheme, we have to specify how the *quality* of such a scheme should be measured. All considered width parameters are *homogeneous* in the sense that each separator is measured by the same criterion, depending only on the width parameter, e.g., measuring the number of vertices leads to the parameter *treewidth*, whereas measuring how easily an edge separator can be constructed from bicliques[3] yields the parameter *clique-width*. The *width* of the decomposition is then determined by the maximum measure attained by any occurring separator. Intuitively, a graph has small treewidth if it can be recursively decomposed by only using small vertex separators and, similarly, a graph has small clique-width if it can be recursively decomposed by densely structured edge

[3] A *biclique*, or *complete bipartite graph*, is a graph whose vertices can be partitioned into two sides such that every edge between the two sides exists, but no other edges exist. Therefore a biclique captures a simple kind of dense structure.

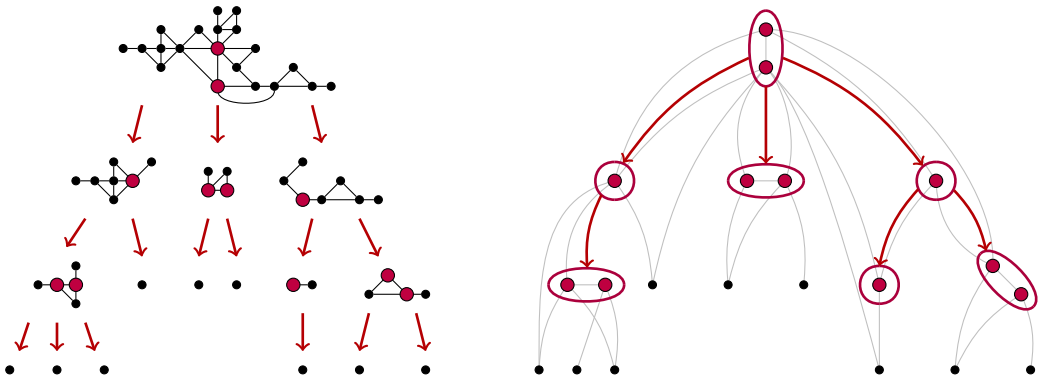


Fig. 1.2.: On the left side, a graph is recursively decomposed in a divide-and-conquer manner by repeatedly removing small vertex separators, which are denoted by the red vertices in each step, and each red arrow corresponds to a connected component arising from the separator removal. On the right side, we arrange the separators used in the decomposition on the left in a hierarchical decomposition scheme of the graph. Each red-encircled group of vertices corresponds to a separator used in a decomposition step and the red arrows indicate the hierarchy of separators. The edges of the original graph are hinted at in light gray; due to the separator structure, the edges outside of separators can only connect ancestors and descendants.

separators.[4] A particular consequence of the width definition is that a decomposition of low width allows for arbitrarily many disjoint separators with the same quality, e.g., size, forming a thick path-like graph, see Figure 1.3.

Dynamic Programming for Width Parameters. Equipped with a hierarchical decomposition of low width, we can solve many problems on the given graph. Motivated by its efficacy on trees, the simplest hierarchical structures, we solve graph problems via bottom-up *dynamic programming* along the hierarchical decomposition and build up solutions for the considered problem by combining partial solutions from each piece at the chosen separators. Whether two partial solutions can be combined at a separator is determined by their respective interaction with the separators; this interaction will be referred to as the *state* of a partial solution throughout this thesis. The guarantee of low width allows us to conclude for appropriate graph problems that there are only a few distinct states, thus fast dynamic programming algorithms can be obtained by solving a subproblem for each possible state on a separator.

Number of States. The number of states is crucial for the running time of the considered dynamic programming algorithms as the number of considered subproblems scales with it. Indeed, for the studied single-exponential algorithms with running time $\mathcal{O}^*(\alpha^k)$ when given a decomposition of width k, the running time, except for the polynomial part, can be explained as follows: The states are determined by essentially making one of α choices for each unit of width, e.g., in the case of treewidth there are α choices per vertex in the

[4] The formal definitions of treewidth and clique-width implicitly impose additional restrictions, which are not captured by our informal discussion, on how different separators must be related.

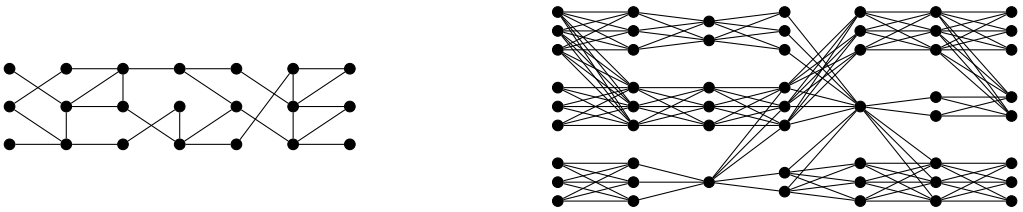


Fig. 1.3.: The left graph shows a path-like graph of treewidth 3, where every column of vertices is a vertex separator. The right graph shows a path-like graph of clique-width 4, where every column of edges is a structured edge separator in the sense of clique-width.

separator, hence leading to α^k subproblems for a separator of size k. Thus, the base α of the running time also conveys structural information about the problem, as it captures how complicated the interaction between the problem and the considered separators is. In particular, by ruling out smaller running time bases, we also learn that the considered states are essentially the correct ones. For these reasons, we conduct a precise study of how many states, and which ones, can be attained at the different separator types for various graph problems.

Dense Width Parameters. Many tight fine-grained results are known relative to treewidth [27, 45, 48, 51, 59, 60, 63, 68, 71, 72, 115, 126, 131, 132, 145, 146], thus treewidth serves as a good starting point for fine-grained investigations relative to width parameters. However, far fewer tight results are known relative to dense width parameters [86, 106, 108, 114, 123], as optimizing the number of states relative to densely structured separators is more challenging than relative to small vertex separators. In particular, applying the standard lower bound technique relative to width parameters of Lokshtanov et al. [126] to dense width parameters requires carefully selecting a specific subset of states. Instead, simply selecting all possible states is often appropriate for treewidth. Based on the tight results of Cygan et al. [50, 51] for connectivity problems parameterized by treewidth, we explore their fine-grained complexity parameterized by the dense width parameters *clique-width* and *modular-treewidth*.

Clique-Width. Coming from the theory of graph grammars, see e.g. the book of Courcelle and Engelfriet [41], clique-width is defined via algebraic expressions that construct graphs by repeatedly adding bicliques. Clique-width is arguably the most popular dense graph parameter. In particular, all of the cited tight fine-grained results [86, 106, 108, 114, 123] relative to dense width parameters consider (linear-)clique-width. This simultaneously shows that there are still many gaps regarding the fine-grained complexity of problems relative to clique-width, but also gives reason to believe that tight results for connectivity problems parameterized by clique-width are attainable. A disadvantage of clique-width is that the complexity of computing clique-width is not well-understood, in particular, it is even open whether graphs of clique-width 4 can be recognized in polynomial time. The best known algorithms compute an exponential approximation of clique-width in single-exponential time [147] or double-exponential time [74].

Modular-Treewidth. The dense width parameter modular-treewidth combines the *modular decomposition*, which recursively partitions a graph into *modules* whose members have the same neighborhood outside of the module, of a graph with the parameter treewidth. As the modular decomposition can be efficiently computed, see e.g. Tedder et al. [169], the algorithms for computing treewidth essentially transfer to modular-treewidth, yielding e.g. a constant-factor approximation of modular-treewidth in single-exponential time. Modular-treewidth has been studied much less than clique-width; we are only aware of the work by Bodlaender and Jansen [22] on MAXIMUM CUT and some works on a related, but more restrictive parameter [123, 133, 150]. However, modular-treewidth can be viewed as a natural intermediate step between treewidth and clique-width, thus we can hope for improved algorithms compared to clique-width. Moreover, the connection between treewidth and modular-treewidth leads to a more direct transfer of techniques, thus allowing us to obtain a greater number of tight results.

1.1.1 A Case Study: 3-COLORING

To illustrate how the number of states can be influenced by the considered separator type, we study the 3-COLORING problem. In the 3-COLORING problem, we are given a graph $G = (V, E)$ with vertex set V and edge set E, and we must decide whether it is possible to partition the vertices into three disjoint sets $V = V_1 \cup V_2 \cup V_3$ such that each of them is an *independent set*, i.e., no edge of G is fully contained in any one of these sets. For our sake, the formalism of 3-coloring functions is more convenient; we say that a function $\varphi\colon V \to \{1, 2, 3\}$ is a 3-*coloring* of G if $\varphi(u) \neq \varphi(v)$ for every edge $\{u, v\} \in E$, capturing that the endpoints of each edge must be assigned to different sets. With this formalism, the task is to decide if a 3-coloring φ of G exists.

Abstractly, given a separator S that decomposes the graph into two pieces A and B, we want to know how much information about the interaction with S is necessary for us to decide whether two partial solutions, one living on A and one living on B, combine to a solution of the whole graph G. This can be understood in terms of *communication complexity* as follows. Suppose that Alice has a partial solution φ_A living on A and Bob has a partial solution φ_B living on B. How much information must Alice and Bob communicate, so that they can decide whether the combination of φ_A and φ_B yields a 3-coloring of G?

Small Vertex Separator

We first consider a small vertex separator, which corresponds to the parameters pathwidth and treewidth. A *separation* of G is a pair (A, B) of vertex sets such that $V = A \cup B$ and such that there are no edges between $A \setminus B$ and $B \setminus A$, in other words, $S = A \cap B$ is a vertex separator of G. Having fixed a separation, a partial solution on A is simply a 3-coloring $\varphi_A\colon A \to \{1, 2, 3\}$ of $G[A]$, i.e., the graph induced by A, and similarly for a partial solution on B. Since each part is correctly colored by the corresponding partial solution and since there are no edges between $A \setminus B$ and $B \setminus A$, the partial solutions φ_A and φ_B can be combined into a 3-coloring of G if they agree on the separator $S = A \cap B$, i.e., $\varphi_A\big|_{A\cap B} = \varphi_B\big|_{A\cap B}$. Formally, we denote the resulting 3-coloring by $\varphi_A \oplus \varphi_B\colon V \to \{1, 2, 3\}$,

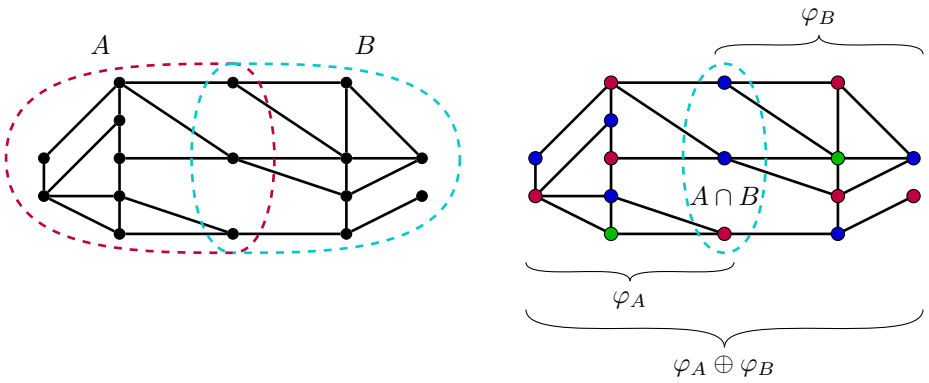


Fig. 1.4.: The left graph shows a separation (A, B) with a vertex separator $S = A \cap B$ of size 3. The right graph shows how a 3-coloring φ_A of $G[A]$ and a 3-coloring φ_B of $G[B]$ that agree on $S = A \cap B$ combine to a 3-coloring $\varphi_A \oplus \varphi_B$ of G.

where $(\varphi_A \oplus \varphi_B)(v) = \varphi_A(v)$ for $v \in A$ and $(\varphi_A \oplus \varphi_B)(v) = \varphi_B(v)$ for $v \in B$; this is well-defined due to the condition $\varphi_A\big|_{A\cap B} = \varphi_B\big|_{A\cap B}$, see Figure 1.4 for an example.

Returning to the communication complexity perspective, we therefore see that it suffices for Alice to simply communicate the function $\varphi_A\big|_{A\cap B}$. If $A \cap B$ consists of k vertices, then there are 3^k possible functions from $A \cap B$ to $\{1, 2, 3\}$, which correspond to the possible states. By prescribing a function $f\colon A \cap B \to \{1, 2, 3\}$, we obtain the subproblem on A (and analogously on B) that asks if there is 3-coloring of $G[A]$ that agrees with f on $A \cap B$. This idea can be turned into an $\mathcal{O}^*(3^k)$-time dynamic programming algorithm for 3-Coloring on graphs of treewidth k. Assuming the Strong Exponential-Time Hypothesis (SETH), Lokshtanov et al. [126] have shown that the base 3 is optimal.

Dense Edge Separator

Next, we consider a densely structured edge separator, but we first need some preliminary definitions to make the structure precise. Suppose we have two disjoint vertex sets A and B with $V = A \cup B$ that are additionally *k-labeled*, i.e., there are functions $\mathtt{lab}_A\colon A \to [k] = \{1, 2, \ldots, k\}$ and $\mathtt{lab}_B\colon B \to [k]$ giving one of k labels to each vertex of A (or B). The edge separator is described by a relation $R \subseteq [k] \times [k]$ such that for all $a \in A$ and $b \in B$ we have that $\{a, b\} \in E$ if and only if $(\mathtt{lab}_A(a), \mathtt{lab}_B(b)) \in R$. Therefore, fixing two labels ℓ_A and ℓ_B, either no edges exist between $\mathtt{lab}_A^{-1}(\ell_A)$ and $\mathtt{lab}_B^{-1}(\ell_B)$ or all edges between them exist, i.e., the edges between them induce a biclique with sides $\mathtt{lab}_A^{-1}(\ell_A)$ and $\mathtt{lab}_B^{-1}(\ell_B)$, see Figure 1.5 for an example. Hence, if k is small, then the considered separator can be easily constructed by adding bicliques that respect the labels on each side. Such separators correspond to the parameter clique-width[5].

Given a 3-coloring φ_A of $G[A]$ and a 3-coloring φ_B of $G[B]$, we again ask how much information is necessary to determine whether $\varphi_A \oplus \varphi_B$, where the previous condition $\varphi_A\big|_{A\cap B} = \varphi_B\big|_{A\cap B}$ is now trivial due to $A \cap B = \emptyset$, yields a 3-coloring of G. By considering

[5]However, how we have defined them here is closer to the related parameter *nlc-width*, see Wanke [173] and Johansson [112].

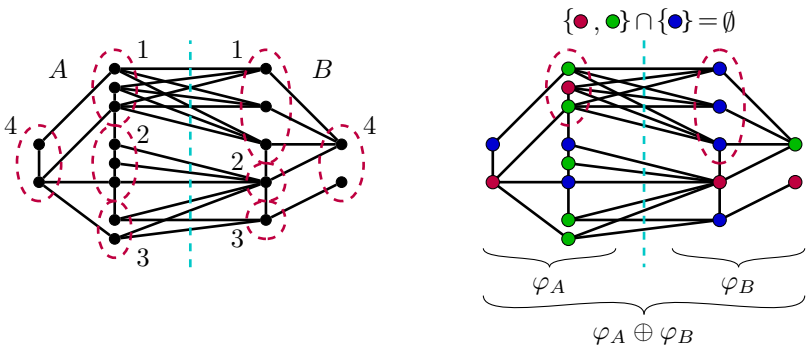


Fig. 1.5.: The left graph is partitioned into two disjoint vertex sets A and B that are additionally 4-labeled; all vertices in the same red-encircled group have the same label, which is noted next to the group. The edge separator between A and B in the left graph can be described by the relation $R = \{(1,1),(2,2),(3,2),(3,3)\}$. The right graph depicts a 3-coloring φ_A of $G[A]$ and a 3-coloring φ_B of $G[B]$ that combine to a 3-coloring $\varphi_A \oplus \varphi_B$ of G. We exemplify Lemma 1.1.1 for the pair $(1,1) \in R$ with the right graph. Since $\varphi_A \oplus \varphi_B$ is a 3-coloring of G, the color set $\varphi_A(\mathtt{lab}_A^{-1}(1))$ attained by vertices in A with label 1 is disjoint from the color set $\varphi_B(\mathtt{lab}_B^{-1}(1))$ attained by vertices in B with label 1.

for a pair $(\ell_A, \ell_B) \in R$, the set of colors $\varphi_A(\mathtt{lab}_A^{-1}(\ell_A))$ used on vertices with label ℓ_A and analogously for label ℓ_B, we can make the following observation, see also Figure 1.5.

Lemma 1.1.1. *Under the aforementioned conditions, $\varphi := \varphi_A \oplus \varphi_B$ is a 3-coloring of G if and only if $\varphi_A(\mathtt{lab}_A^{-1}(\ell_A)) \cap \varphi_B(\mathtt{lab}_B^{-1}(\ell_B)) = \emptyset$ for every pair $(\ell_A, \ell_B) \in R$.*

Proof. If φ is 3-coloring of G and $(\ell_A, \ell_B) \in R$, then the two color sets $\varphi_A(\mathtt{lab}_A^{-1}(\ell_A))$ and $\varphi_B(\mathtt{lab}_B^{-1}(\ell_B))$ must be disjoint: Otherwise, there is an edge between $\mathtt{lab}_A^{-1}(\ell_A)$ and $\mathtt{lab}_B^{-1}(\ell_B)$ whose endpoints get the same color, as all edges between these two vertex sets exist by definition of R.

If φ is not a 3-coloring of G, then this can only be because some edge $\{a,b\} \in E$, $a \in A$, $b \in B$, between A and B is incorrectly colored, i.e., $\varphi(a) = \varphi(b)$. Let $\ell_A = \mathtt{lab}_A(a)$ and $\ell_B = \mathtt{lab}_B(b)$. By definition of R, we have that $(\ell_A, \ell_B) \in R$. However, $\varphi(a) = \varphi(b)$ implies that $\varphi_A(\mathtt{lab}_A^{-1}(\ell_A)) \cap \varphi_B(\mathtt{lab}_B^{-1}(\ell_B)) \neq \emptyset$. □

Lemma 1.1.1 motivates that Alice should communicate the color sets $\varphi_A(\mathtt{lab}_A^{-1}(\ell_A))$ for all $\ell_A \in [k]$ instead of the precise mapping φ_A. Since there are 3 possible colors, there must be $2^3 = 8$ possible color sets, thus there are 8 possibilities per label. However, this can be still optimized. Clearly, unused labels, i.e., $\mathtt{lab}_A^{-1}(\ell_A) = \emptyset$, can be ignored, and for all used labels we have that $\varphi_A(\mathtt{lab}_A^{-1}(\ell_A)) \neq \emptyset$, so that the empty color set can be eliminated from the possibilities. Furthermore, we only need to consider labels ℓ_A such that there exists an ℓ_B with $(\ell_A, \ell_B) \in R$, otherwise the vertices in $\mathtt{lab}_A^{-1}(\ell_A)$ are not incident to any edge of the edge separator. However, for such labels ℓ_A, we can eliminate the color set $\{1,2,3\}$ as we then have $\varphi_A(\mathtt{lab}_A^{-1}(\ell_A)) \cap \varphi_B(\mathtt{lab}_B^{-1}(\ell_B)) \neq \emptyset$ for every 3-coloring φ_B of $G[B]$. Thus, we have reduced the possibilities to 6 per label. This idea leads to an $\mathcal{O}^*(6^k)$-time algorithm for 3-Coloring on graphs of clique-width k and Lampis [123] has shown that the base 6 is optimal, assuming SETH.

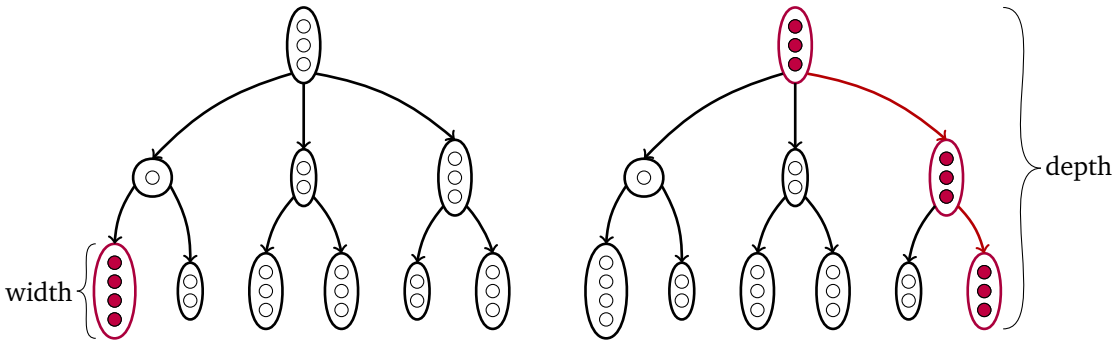


Fig. 1.6.: A depiction of the same hierarchical decomposition scheme twice, where the graph edges and the remaining pieces after the final decomposition round are omitted for readability's sake. This decomposition scheme uses small vertex separators. On the left side, we measure its width at a largest vertex separator denoted in red. On the right side, we measure its depth at a root-leaf path, denoted in red, leading to the maximum sum of separator sizes.

1.2 Depth Parameters and Modulators

Width parameters restrict the complexity of any single separator in a hierarchical decomposition scheme, but not how separators are allowed to interact or how many separators there are. Such restrictions further constrain the allowed graph structure and it is natural to ask whether this allows for algorithmic improvements compared to width parameters. We will now go beyond width parameters and study *depth parameters* and *modulators*, which can capture the mentioned restrictions and do not fall under the framework of width parameters.

From Hierarchical Decomposition Scheme to Depth Parameters. Depth parameters are derived from the same hierarchical decomposition schemes as width parameters, but the mechanism by which the quality of such a scheme is evaluated is different. Whereas the width of a decomposition scheme is given by the maximum measure attained by any occurring separator, the *depth* of a decomposition scheme is given by *accumulating* the separator measures along each root-leaf-path in the decomposition scheme and taking the maximum of these accumulated measures, see Figure 1.6. Thus, the depth of a decomposition scheme bounds the number of decomposition rounds, i.e., how often we are allowed to recurse, but also accounts for the complexity of the separators used, e.g., after using a complex separator only a few additional decomposition rounds or simple separators are allowed. In particular, a decomposition scheme of low depth does not allow for arbitrarily long path-like graphs, where we have many disjoint separators with the same measure.

Branching Algorithms. A hierarchical decomposition scheme of low depth can be used to guide a *branching algorithm* as follows. Starting at the topmost separator, we make a number of branching decisions depending on the measure of the separator and then proceed downwards, recursing independently on each graph piece resulting from the separator removal. After the final decomposition round, only a sufficiently simple graph remains, where the subproblem can be solved directly. For such a branching algorithm, the maximum

number of branching decisions is bounded by the depth of the hierarchical decomposition scheme. Thus, whereas width parameters model dynamic programming, depth parameters naturally model *branching algorithms*. Since fast dynamic programming algorithms usually achieve their speed by trading away space, they tend to have large space requirements, which is in particular true for the considered dynamic programming algorithms relative to width parameters. We will see that in many cases branching algorithms relative to depth parameters with far smaller space requirements, but similar time complexity, can be obtained.

Treedepth. The space of depth parameters is far less explored than the space of width parameters. Even for treedepth, the depth analog of treewidth, and thus arguably the best-understood depth parameter, many optimal bases remain unknown, e.g., the optimal base for DOMINATING SET. Many dynamic programming algorithms parameterized by treewidth can be transformed into branching algorithms parameterized by treedepth, while essentially maintaining the base of the running time but drastically reducing the space requirement [37, 80, 155]. Although the required transformations can be nontrivial, we show that this phenomenon extends to many connectivity problems. As it is very challenging to transfer the lower bound techniques relative to width parameters over to depth parameters, we are unable to complement the branching algorithms for connectivity problems parameterized by treedepth with tight lower bounds certifying the optimality of the obtained bases. The quest for tight lower bounds relative to depth parameters naturally leads to the concept of modulators.

Modulators. Modulators describe the extreme case where the hierarchical decomposition scheme consists of only a single separator, so that after one decomposition step all remaining graph pieces must be sufficiently simple. Hence, admitting a small or simply structured modulator is typically a far more restrictive requirement than admitting a hierarchical decomposition scheme of low width or even low depth, however in general it depends on when a graph is considered to be simple. In particular, lower bounds relative to appropriate modulators transfer to depth and width parameters. Unlike width and depth parameters, modulators are not quite as directly associated with an algorithmic paradigm. However, parameterizing by modulator size often allows for good kernelization[6] guarantees and the intersection of a solution with the modulator can be completely guessed, e.g., for many benchmark problems the parameterization by solution size is a modulator-parameterization, which greatly facilitates the technique of *iterative compression*, see e.g., the book of Cygan et al. [47].

Lower Bounds Relative to Modulators. As modulators allow for a much larger breadth of algorithmic paradigms than width parameters, it is natural to expect that considerable speed-ups can be obtained. It is known that for homomorphism problems, which generalize coloring problems and do not involve any cardinality or cost constraint, such speed-ups are not possible [109, 126, 151]. By providing novel lower bound constructions, we show

[6] A *kernelization* is a polynomial-time preprocessing algorithm for parameterized problems that reduces an instance down to an instance whose size is bounded by a function $g(k)$ in the parameter k and preserves the answer.

that also for coloring-related vertex-deletion problems such speed-ups are impossible, and already a single complex separator is sufficient to cause the same complexity as in the general width parameter case. The considered modulators for these lower bounds are modulators to constant treewidth, i.e., after removing the modulator a graph of constant treewidth remains, and a dense analog of such modulators. These modulators forbid multiple disjoint balanced separators, but still allow for long paths, which gives us enough flexibility to design the lower bound constructions. Not only do these lower bounds aid our understanding of which structural features are algorithmically difficult, but they also witness that the branching algorithms relative to depth parameters cannot improve the base beyond what is achieved by dynamic programming relative to width parameters in these cases.

1.3 Connectivity Problems

As connectivity problems are one of the main protagonists in this thesis, we dedicate a separate introductory section to them. Connectivity constraints are *global constraints* as opposed to locally checkable constraints, therefore it is often more challenging to design algorithms for connectivity problems. We consider essentially three types of connectivity problems in this thesis. The first type is the "pure" connectivity problem Steiner Tree, where a set of vertices K, called *terminals*, must be connected as cheaply as possible without any further constraints. The second type consists of locally checkable graph problems that have been augmented by a connectivity constraint, such as Connected Vertex Cover and Connected Dominating Set, where we have to find a *connected* solution to the original problem. The requirement that solutions induce a connected graph can be viewed as a *positive* connectivity constraint. As the final type, we consider Feedback Vertex Set which exemplifies a *negative* connectivity constraint, where we have to ensure that only an acyclic graph remains after deleting the vertices in a solution, i.e., all 2-connected subgraphs must be destroyed.

Treewidth. Cygan et al. [50, 51] obtain optimal fine-grained results for connectivity problems parameterized by treewidth. Their algorithms use the *cut-and-count-technique* which transforms the connectivity constraint to a locally checkable constraint plus modulo-2 counting, thus making connectivity problems more amenable to dynamic programming techniques. The lower bounds are obtained by the approach of Lokshtanov et al. [126] and handling the connectivity constraint by checking the connectivity to a distinguished vertex. As each of these techniques uses a subtly different set of states, the study of the fine-grained parameterized complexity of connectivity problems is particularly interesting.

Dense Width Parameters. Results of Bergougnoux and Kanté [12] show that single-exponential algorithms also exist for many connectivity problems parameterized by dense width parameters, however the obtained bases are not optimal. Motivated by the success of the cut-and-count-technique for treewidth, we apply the cut-and-count-technique relative to dense width parameters and combine it with problem-specific insights to obtain optimal

bases. In particular, we obtain for the parameter sequence treewidth, modular-treewidth, clique-width that the optimal base for

- CONNECTED VERTEX COVER starts at 3, increases to 5, and then increases to 6, and for
- CONNECTED DOMINATING SET starts at 4, stays at 4, and then increases to 5.

These two sequences of optimal bases show for dense width parameters that the combination of connectivity constraint and standard problem constraints can interact very differently with the considered separators. It is also interesting to compare with VERTEX COVER, which has optimal base 2 relative to all three parameters, showing that the impact of the connectivity constraint can get worse for more expressive width parameters.

Treedepth. The dynamic programming algorithms relative to treewidth of Cygan et al. [51] imply the same running times for connectivity problems relative to treedepth. As discussed, we are interested in transforming these dynamic programming algorithms into branching algorithms relative to treedepth while maintaining the base of the running time but reducing the space requirement. The presence of the global connectivity constraint makes this seem like a daunting task at first glance. Fortunately, the constraint transformation of the cut-and-count-technique facilitates the transformation into a branching algorithm. Thus, the cut-and-count-technique allows us to obtain, in particular, polynomial-space branching algorithms with running time $\mathcal{O}^*(3^k)$ and $\mathcal{O}^*(4^k)$ for CONNECTED VERTEX COVER and CONNECTED DOMINATING SET parameterized by treedepth, respectively.

1.4 Organization of the Thesis

This thesis is structured in 4 parts. In Part I, we first discuss the preliminary definitions in Chapter 2, which in particular contains a detailed look at the considered graph parameters. In Chapter 3, we give an introduction to two central techniques for this thesis; the *cut-and-count-technique* to obtain algorithms for *connectivity problems*, and the general *lower bound principle* to obtain fine-grained lower bounds relative to width parameters.

In Part II, we study several benchmark connectivity problems relative to the two dense width parameters *clique-width* and *modular-treewidth*. We design novel single-exponential algorithms using the cut-and-count-technique relative to these parameters and in many cases show that the obtained running time base is optimal. See Section 4.5 for a more detailed discussion of the organization of Part II. The results of Part II are based on the preprints [100] and [101], which are both joint work with Stefan Kratsch.

In Part III, we go beyond width parameters and investigate what algorithmic improvements are possible for depth parameters and modulators. First, we obtain fast and space-efficient branching algorithms for multiple connectivity problems relative to *treedepth*. Secondly, we consider a class of vertex-deletion problems related to q-COLORING and show that they do not admit improved algorithms relative to modulators by designing novel lower bounds that diverge from the construction principle for width parameters. We refer to Section 10.3 for a thorough discussion of how Part III is organized. The results of Part III are based on the articles [99] and [102], which are both joint work with Stefan Kratsch.

We conclude in Part IV and present future work.

Part I

Basics

2 Preliminaries

We discuss the preliminaries in this chapter. Some basic notation is defined Section 2.1. In Section 2.2 and Section 2.3, we discuss the necessary basics of classical complexity theory and parameterized complexity theory. In Section 2.4, we formally define the considered graph parameters and study the relationships among them.

The problem definitions can be found in appendix A.1.

2.1 Notation

Miscellaneous Notation. For $n_1, n_2 \in \mathbb{Z}$, we write $[n_1, n_2] = \{x \in \mathbb{Z} : n_1 \leq x \leq n_2\}$ and $[n_2] = [1, n_2]$. For a boolean predicate p, we let $[p]$ denote the *Iverson bracket* of p, which is 1 if p is true and 0 if p is false.

For two integers a, b we write $a \equiv_c b$ to indicate equality modulo $c \in \mathbb{N}$, i.e., $a \equiv_c b$ if and only if $a - b$ is divisible by c. For $c \in \mathbb{N}_{\geq 2}$, we denote by $\mathbb{Z}_c$ we denote the ring of integers modulo c. For a field or ring $\mathbb{F}$ we denote by $\mathbb{F}[Z_1, Z_2, \ldots, Z_t]$ the ring of polynomials in the indeterminates $Z_1, Z_2, \ldots, Z_t$ with coefficients in $\mathbb{F}$.

The *power set* of a set A is denoted by $\mathcal{P}(A)$. For a set S and $0 \leq k \leq |S|$, we define $\binom{S}{k} = \{T \subseteq S : |T| = k\}$, and $\binom{S}{\leq k} = \{T \subseteq S : |T| \leq k\}$, and $\binom{S}{\geq k}$ analogously. For $0 \leq k \leq n$, we define $\binom{n}{\leq k} = |\binom{[n]}{\leq k}|$ and similarly $\binom{n}{\geq k}$. For a set family $\mathcal{S}$, we define $\bigcup(\mathcal{S}) = \bigcup_{S \in \mathcal{S}} S$. A set family $\mathcal{F}$ is called *laminar* if for every $X, Y \in \mathcal{F}$ it holds that $X \cap Y = \emptyset$, $X \subseteq Y$, or $Y \subseteq X$. An ordered tuple of sets $(A_1, \ldots, A_\ell)$ is an *ordered subpartition* if $A_i \cap A_j = \emptyset$ for all $i \neq j \in [\ell]$.

Function Notation. Given a function $f \colon A \to B$, we denote its *domain* by $\mathtt{dom}(f) = A$. For a function $f \colon V \to \mathbb{Z}$ and a subset $W \subseteq V$, we write $f(W) = \sum_{v \in W} f(v)$. For functions $g \colon A \to B$, where $B \not\subseteq \mathbb{Z}$, and $A' \subseteq A$, we still denote the *image of A' under g* by $g(A') = \{g(v) : v \in A'\}$. If $f \colon A \to B$ is a function and $A' \subseteq A$, then $f\big|_{A'}$ denotes the *restriction* of f to A'; and for a subset $B' \subseteq B$, we denote the *preimage of B' under f* by $f^{-1}(B') = \{a \in A : f(a) \in B'\}$. For a function f we denote by $f[v \mapsto \alpha]$ the function $(f \setminus \{(v, f(v))\}) \cup \{(v, \alpha)\}$, viewing f as a set; we also write $f[v \mapsto \alpha, w \mapsto \beta]$ instead of $(f[v \mapsto \alpha])[w \mapsto \beta]$. For two functions $f, g \colon A \to \mathbb{N}$, we write $f(n) = \mathcal{O}(g(n))$ if there exist constants n_0 and c such that $f(n) \leq cg(n)$ for all $n \geq n_0$. When discussing the running time of algorithms, we use the $\mathcal{O}^*$-notation to hide polynomial factors in the input size.

Graph Notation. We use common graph-theoretic notation, mostly following the book of Diestel [55]. Let $G = (V, E)$ be an undirected graph. We denote the number of vertices by n and the number of edges by m. Furthermore, we denote by $\mathtt{cc}(G)$ the number of connected

components of G. For a vertex set $X \subseteq V$, we denote by $G[X]$ the subgraph of G that is induced by X, and $G - X$ denotes the graph $G[V \setminus X]$. The *complement graph* of G is given by $\overline{G} = (V, \binom{V}{2} \setminus E)$.

The *open neighborhood* of a vertex v is given by $N_G(v) = \{u \in V : \{u, v\} \in E\}$, whereas the *closed neighborhood* is given by $N_G[v] = N_G(v) \cup \{v\}$. We extend these notations to sets $X \subseteq V$ by setting $N_G[X] = \bigcup_{v \in X} N_G[v]$ and $N_G(X) = N_G[X] \setminus X$. The degree of $v \in V$ is denoted by $\deg_G(v) = |N_G(v)|$. For two disjoint vertex sets $A, B \subseteq V$, we let $E_G(A, B) = \{\{a, b\} \in E(G) : a \in A, b \in B\}$ denote the set of edges with one endpoint in A and the other in B. Furthermore, adding a *join* between A and B means adding an edge between every vertex in A and every vertex in B. For a vertex set $X \subseteq V$, we define $\delta_G(X) = E_G(X, V \setminus X)$ and we write $\delta_G(v) = \delta_G(\{v\})$ for single vertices v. When the underlying graph is clear from the context, we omit the subscript in these notations.

A separation of G is a pair of vertex subsets (A, B) with $A \cup B = V$ such that $E(A \setminus B, B \setminus A) = \emptyset$; the set $A \cap B$ is the *(vertex) separator* of this separation. An *edge separator* is an edge set $E' \subseteq E$ whose removal increases the number of connected components of G.

A *cut* of a set $X \subseteq V$ is a pair (X_L, X_R) with $X_L \cap X_R = \emptyset$ and $X_L \cup X_R = X$, we also use the notation $X = X_L \uplus X_R$. We refer to X_L and X_R as the *left side* and *right side* of the cut, respectively. Note that either side may be empty, although in our applications the left side is usually nonempty. A *cut* of a graph $G = (V, E)$ is a cut of the vertex set V.

An *r-coloring* of a graph $G = (V, E)$ is a function $\varphi \colon V \to [r]$ such that $\varphi(u) \neq \varphi(v)$ for all $\{u, v\} \in E$. We say that G is *r-colorable* if there is an r-coloring of G; G is *bipartite* if G is 2-colorable. The *chromatic number* of G, denoted by $\chi(G)$, is the minimum r such that G is r-colorable. A vertex set $X \subseteq V$ is an *independent set* of G if $G[X]$ contains no edges. A vertex set $X \subseteq V$ is a *vertex cover* of G if $G - X$ contains no edges. A vertex set $X \subseteq V$ is a *clique* of G if $G[X]$ is a complete graph. A vertex set $X \subseteq V$ is a *biclique* if $G[X]$ is a complete bipartite graph. A *vertex-disjoint packing* $\mathcal{P}$ in G is a collection of subgraphs of G such that no pair of subgraphs share a vertex. A *matching* is a vertex-disjoint packing of edges.

Satisfiability. A *literal* is a boolean variable x or its negation $\neg x$. A *clause* is a finite set of literals. A boolean formula σ in *conjunctive normal form (CNF)* is a finite set of clauses; the set of variables occurring in σ is denoted $\mathrm{var}(\sigma)$. A *(partial) truth assignment* of σ is a function $\tau \colon X \to \{0, 1\}$, where $X \subseteq \mathrm{var}(\sigma)$; we say that τ *satisfies* σ if in every clause of σ there is some positive literal x with $\tau(x) = 1$ or some negative literal $\neg x$ with $\tau(x) = 0$. We say that σ is *satisfiable* if a truth assignment $\tau \colon \mathrm{var}(\sigma) \to \{0, 1\}$ exists that satisfies σ.

Strong Exponential-Time Hypothesis. The *Strong Exponential-Time Hypothesis* (SETH) [35, 105] concerns the complexity of q-SATISFIABILITY. We define for all $q \geq 3$ the quantity

$$c_q = \inf\{\delta : q\text{-SATISFIABILITY can be solved in time } \mathcal{O}(2^{\delta n})\}.$$

The weaker *Exponential-Time Hypothesis* (ETH) of Impagliazzo and Paturi [104] posits that $c_3 > 0$, whereas the Strong Exponential-Time Hypothesis states that $\lim_{q \to \infty} c_q = 1$. Or equivalently, for every $\delta < 1$, there is some q such that q-SATISFIABILITY cannot be solved in

time $\mathcal{O}(2^{\delta n})$. Our lower bounds will rely on SETH, as ETH is not strong enough to allow us to pin down the running time base of single-exponential algorithms. However for some of our lower bounds, the following weaker variant of SETH, also called CNF-SETH, see e.g. the survey of Lokshtanov et al. [127], is sufficient.

Conjecture 2.1.1 (CNF-SETH). *For every $\varepsilon > 0$, there is no algorithm solving* SATISFIABILITY *with n variables and m clauses in time $\mathcal{O}(\text{poly}(m)(2-\varepsilon)^n)$.*

In some cases, we make use of the following equivalent statements.

Theorem 2.1.2 ([46]). *The following statements are equivalent to SETH:*

1. *For all $\delta < 1$, there is a clause size q such that q-*SATISFIABILITY *cannot be solved in time $\mathcal{O}(2^{\delta n})$, where n is the number of variables.*
2. *For all $\delta < 1$, there is a set size q such that q-*HITTING SET*, i.e., all sets contain at most q elements, cannot be solved in time $\mathcal{O}(2^{\delta n})$, where n is the universe size.*

2.2 Complexity

To formally study the impact of *instance structure* on *problem complexity*, we make use of the language of complexity theory and, in particular, parameterized complexity. We begin by giving a quick overview of the relevant classical complexity theory, for more details we refer to the book of Arora and Barak [5]. All problems considered in this thesis are *decision problems*, i.e., problems for which the only possible answers are "true" and "false", or *acceptance* and *rejection* respectively. Formally, decision problems are modeled as *languages* $L \subseteq \Sigma^*$, where Σ is some finite *alphabet* and Σ^* is the set of all *finite strings* over Σ, and the goal is to decide for a given *instance* $x \in \Sigma^*$ whether x is contained in L. If $x \in L$, we say that x is a *positive instance* or *yes-instance*, and otherwise x is a *negative instance* or *no-instance*.

Typically, the considered decision problems arise from some underlying *optimization problem*, where we have to find given an instance $x \in \Sigma^*$ a *feasible solution* $y \in \mathcal{S}(x)$, where $\mathcal{S}(x)$ is a set containing all feasible solutions of x, optimizing some *objective function* $\mathbf{obj}_x \colon \mathcal{S}(x) \to \mathbb{Z}$. By additionally supplying an instance x with a *target value* $\bar{b} \in \mathbb{Z}$, often also called *budget*, we can turn this into a decision problem as follows. Given a pair $(x, \bar{b})$, which can also be modeled as a string, we have to decide whether there exists a feasible solution $y \in \mathcal{S}(x)$ with $\mathbf{obj}_x(y) \leq \bar{b}$ if we considered a *minimization problem* or $\mathbf{obj}_x(y) \geq \bar{b}$ if we considered a *maximization problem*.

Complexity Classes. We can arrange decision problems into *complexity classes* based on which kind of algorithms they admit. In classical complexity theory, we measure the running time of an algorithm in terms of the *input size*[1] $|x|$. The complexity class P consists of all decision problems L that can be *solved* in polynomial time, i.e., there exists a polynomial-time algorithm that correctly accepts or rejects based on whether the given instance x is

[1] For graph problems, we usually do not directly measure in terms of the input size, but instead in terms of the number of vertices n and sometimes the number of edges m. We assume that a reasonable encoding of graphs is used, i.e., a graph with n vertices should be encoded using $\mathcal{O}(n^2)$ bits.

contained in L. The complexity class NP consists of all decision problems L that can be *verified* in polynomial time, i.e., there exists a polynomial-time algorithm $\mathbb{A}$ that gets as input a pair (x, y), where $y \in \Sigma^*$ is a *certificate*, and if $x \in L$, then there exists a certificate y of polynomially-bounded size such that $\mathbb{A}$ accepts (x, y), and if $x \notin L$, then $\mathbb{A}$ rejects (x, y) for all certificates y. Every polynomial-time solvable decision problem can also be verified in polynomial time, therefore $\mathsf{P} \subseteq \mathsf{NP}$.

Reductions. The complexity of problems can be compared via *reductions*. A decision problem $A \subseteq \Sigma^*$ is *polynomial-time (many-one-)reducible to* a decision problem $B \subseteq \Gamma^*$ if there exists a polynomial-time computable function $f\colon \Sigma^* \to \Gamma^*$ such that $x \in A$ if and only if $f(x) \in B$. In that case, A is at least as easy to solve as B and B is at least as hard to solve as A. A decision problem B is NP-*hard* when every decision problem $A \in \mathsf{NP}$ is polynomial-time reducible to B; if additionally $B \in \mathsf{NP}$, then B is NP-*complete*.

P versus NP. The central open question of classical complexity theory is whether $\mathsf{P} = \mathsf{NP}$ or $\mathsf{P} \neq \mathsf{NP}$. The majority of computer scientists believe that $\mathsf{P} \neq \mathsf{NP}$, see the poll conducted by Gasarch [93]. The statement $\mathsf{P} \neq \mathsf{NP}$ in particular implies that NP-hard problems do not admit polynomial-time algorithms. We use $\mathsf{P} \neq \mathsf{NP}$ as one of our complexity hypotheses, since all of our lower bounds assume the much stronger SETH (or CNF-SETH) which implies ETH which in turn implies $\mathsf{P} \neq \mathsf{NP}$. While any of these hypotheses could turn out to be false, the connections between the problem complexities witnessed by our reductions remain.

2.3 Parameterized Complexity

The disadvantage of classical complexity theory is that by measuring the running time only in terms of the input size $|x|$, we cannot study how the structure of x impacts the problem complexity. Let us explain this disadvantage by considering the concept of NP-hardness. When a decision problem $A \subseteq \Sigma^*$ is NP-hard, then it only follows that there is some subfamily of hard, or *worst-case*, instances $I_{hard} \subseteq \Sigma^*$ and for instances $x \in I_{hard}$ it is algorithmically difficult to decide whether $x \in A$ or $x \notin A$. However, A being NP-hard does not imply anything about the complexity of the problem for instances $x \in \Sigma^* \setminus I_{hard}$: they could all be easy to solve, all be hard to solve, or anything in between these two extremes. In particular, this also means that we do not have to give up on solving a problem after its NP-hardness is established. Indeed, the instances we care about could lie outside of I_{hard} and be much easier to solve, but this requires a closer investigation of the problem complexity.

We approach this closer investigation of problem complexity via the framework of parameterized complexity. In essence, the idea of parameterized complexity is to introduce a secondary measure called a *parameter*, usually denoted by k, in addition to the input size $|x|$ and measure the running time of algorithms in terms of both $|x|$ and k. In this thesis, we view the parameter as a quantification of *input structure* in the sense that a lower parameter value represents a more structured instance, however it is also common to parameterize optimization problems by the desired solution size. Whereas the input size $|x|$

is immutable once we have agreed on an encoding, we can consider many different kinds of *parameterizations* that measure various kinds of structure and for each of them investigate its impact on problem complexity. This allows us to understand which kinds of structure make a problem hard and which kinds make it easy. Later, in Section 2.4.4, we also study how parameters, and hence structures, can be comparable with each other and how that affects problem complexity with respect to different parameterizations. We now proceed with the formal definitions, for further details we refer to the books [47, 56, 70, 144].

Definition 2.3.1. A *parameterized problem* is a language $P \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet, and for an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, the value k is called the *parameter*.

Definition 2.3.2. A parameterized problem $P \subseteq \Sigma^* \times \mathbb{N}$ is *fixed-parameter tractable* (FPT) if there exists a computable function $f\colon \mathbb{N} \to \mathbb{N}$, a constant c, and an (FPT-)algorithm $\mathbb{A}$ that on input $(x, k) \in \Sigma^* \times \mathbb{N}$ correctly decides whether $(x, k) \in P$ in time at most $f(k) \cdot |x|^c$. The class of all fixed-parameter tractable problems is denoted FPT.

In most cases, the parameter value k is polynomially bounded by the instance size $|x|$, which implies that the function f must have superpolynomial growth when the underlying decision problem is NP-hard. Therefore, an FPT-algorithm shows that the combinatorial explosion present in the problem can be isolated to only depend on the parameter and not the instance size.

Admitting an FPT-algorithm is essentially the best case for a parameterized problem and the only kind considered in this thesis, however not all parameterized problems admit FPT-algorithms. In the worst case, a parameterized problem could already be NP-hard for constant parameter value, also called *para-NP-hard*, and therefore cannot be in FPT unless P = NP. If instead the problem is polynomial-time solvable for each fixed parameter value, but the degree of the polynomial in the running time is allowed to depend on the parameter value, then we obtain the complexity class XP.

Definition 2.3.3. A parameterized problem $P \subseteq \Sigma^* \times \mathbb{N}$ is *slicewise-polynomial* (XP) if there exist two computable functions $f, g\colon \mathbb{N} \to \mathbb{N}$ and an (XP-)algorithm $\mathbb{A}$ that on input $(x, k) \in \Sigma^* \times \mathbb{N}$ correctly decides whether $(x, k) \in P$ in time at most $f(k) \cdot |x|^{g(k)}$. The class of all slicewise-polynomial problems is denoted XP.

We clearly have that FPT $\subseteq$ XP and it can be seen that FPT $\neq$ XP.

Similar to the theory of NP-hardness, there is also a notion of *parameterized reductions* and the complexity class W[1]. Under the widely believed, see e.g. Downey and Fellows [56], assumption FPT $\neq$ W[1], we can rule out FPT-algorithms by giving a parameterized reduction from a W[1]-hard problem. However, this classification and notion of reductions is far too coarse for our fine-grained setting and hence we omit the formal definitions here. The assumption FPT $\neq$ W[1] is stronger than P $\neq$ NP and it was shown by Chen et al. [36] that ETH implies FPT $\neq$ W[1].

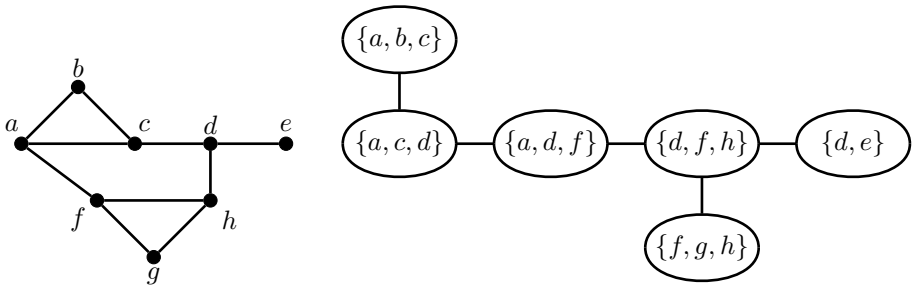


Fig. 2.1.: The right side shows a tree decomposition of width 2 of the graph G on the left side. The vertex sets inside the nodes are the bags of the tree decomposition.

2.4 Graph Parameters

In this section, we give an overview of the main graph parameters considered in this thesis.

A *graph parameter* is a function p that assigns each graph G a number $\mathrm{p}(G) \in \mathbb{N}$. All considered graph parameters are *monotone* wrt. induced subgraphs, i.e., if H is an induced subgraph of G, then $\mathrm{p}(H) \leq \mathrm{p}(G)$.

Parameterized Graph Problems. Let GRAPH PROBLEM denote a *graph problem*, i.e., a decision problem where part of the input is a graph G. Given a graph parameter p, we let GRAPH PROBLEM[p] denote the corresponding parameterized problem where parameter k means that $\mathrm{p}(G) \leq k$. As the considered graph parameters are typically NP-hard to compute, we assume that we are also supplied a *witness* for $\mathrm{p}(G) \leq k$, which is usually some sort of graph decomposition or algebraic expression constructing the graph.

2.4.1 Width Parameters

We begin by giving the formal definitions of the graph parameters considered in this thesis and afterwards, we focus on studying the relationships among them.

Tree Decompositions. A *path/tree decomposition* of a graph $G = (V, E)$ is a pair $(\mathcal{T}, (B_t)_{t \in V(\mathcal{T})})$, where $\mathcal{T}$ is a path/tree and every *bag* $B_t \subseteq V$, $t \in V(\mathcal{T})$, is a set of vertices such that the following properties are satisfied:

- every vertex $v \in V$ is contained in some bag $v \in B_t$,
- every edge $\{v, w\} \in E$ is contained in some bag $\{u, v\} \subseteq B_t$,
- for every vertex v, the set $\{t \in V(\mathcal{T}) : v \in B_t\}$ is connected in $\mathcal{T}$.

The *width* of a path/tree decomposition $(\mathcal{T}, (B_t)_{t \in V(\mathcal{T})})$ is $\max_{t \in V(\mathcal{T})} |B_t| - 1$. The *pathwidth/treewidth* of a graph G, denoted $\mathrm{pw}(G)$ or $\mathrm{tw}(G)$ respectively, is the minimum width of a path/tree decomposition of G.

See Figure 2.1 for an example of a tree decomposition. Intuitively, a tree decomposition of low width shows how a graph can be decomposed by using *small vertex separators*. Even approximating treewidth or pathwidth up to any constant absolute error is NP-hard by Bodlaender et al. [21]. Treewidth can be computed exactly in time $2^{\mathcal{O}(k^3)}n$ by Bodlaen-

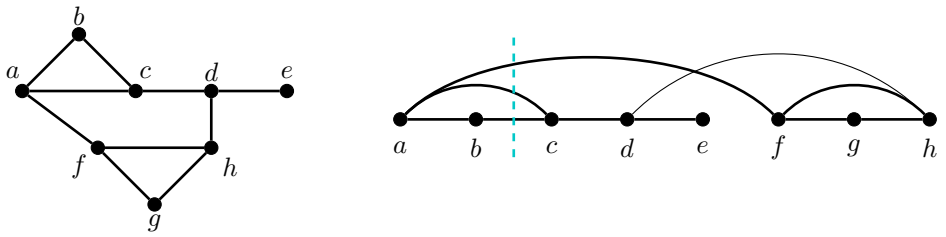


Fig. 2.2.: The right side shows a linear layout of cutwidth 3 of the graph on the left. The dashed cyan line marks a position where cutwidth 3 is attained.

der [17] or in time $2^{\mathcal{O}(k^2)}n^4$ by Korhonen and Lokshtanov [118] or $(2k+1)$-approximated in time $2^{\mathcal{O}(k)}n$ by Korhonen [117]. Pathwidth can be computed exactly in time $2^{\mathcal{O}(k^2)}n$ by Fürer [78].

For dynamic programming algorithms on tree decompositions, we define a standard refinement of tree decompositions that simplifies setting up the various recurrences of the dynamic program. Like Cygan et al. [51], we further refine the *nice tree decompositions* of Kloks [116] and obtain *very nice tree decompositions*.

Definition 2.4.1. A tree decomposition $(\mathcal{T}, (B_t)_{t\in V(\mathcal{T})})$ is *very nice* if it is rooted at the *root node* $\hat{r} \in V(\mathcal{T})$ with $B_{\hat{r}} = \emptyset$ and every bag B_t has one of the following types:

- **Leaf bag:** t has no children and $B_t = \emptyset$.
- **Introduce vertex** v **bag:** t has exactly one child t' and $B_t = B_{t'} \cup \{v\}$ with $v \notin B_{t'}$.
- **Forget vertex** v **bag:** t has one child t' and $B_t = B_{t'} \setminus \{v\}$ with $v \in B_{t'}$.
- **Introduce edge** $\{v, w\}$ **bag:** t is labeled with an edge $\{v, w\} \in E$ and t has one child t' which satisfies $\{v, w\} \subseteq B_t = B_{t'}$.
- **Join bag:** t has exactly two children t_1 and t_2 with $B_t = B_{t_1} = B_{t_2}$.

Furthermore, we require that every edge in E is introduced exactly once.

The following lemma shows that we can always work with very nice tree decompositions after preprocessing.

Lemma 2.4.2 ([51])**.** *Any tree decomposition of G can be converted into a very nice tree decomposition of G with the same width in polynomial time.*

Cutwidth. A *linear layout* of a graph $G = (V, E)$ with n vertices is a linear ordering of its vertices given by a bijection $\pi\colon V \to [n]$. The *cutwidth* of G with respect to π is

$$\mathrm{ctw}_\pi(G) = \max_{i\in[n]} |\{\{u, v\} \in E : \pi(u) \leq i \wedge \pi(v) > i\}|.$$

The *cutwidth* $\mathrm{ctw}(G)$ of G is the minimum cutwidth over all linear layouts of G.

See Figure 2.2 for an example of a linear layout. A linear layout of low width shows how a graph can be decomposed via *small edge separators*. Cutwidth is NP-hard to compute by Garey and Johnson [91] and can be computed exactly in time $2^{\mathcal{O}(k^2)}n$ by Thilikos et al. [171]. As cutwidth does not allow for a tree-like arrangement of the separators, several parameters have been suggested to overcome this issue, such as tree-cut width [176, 89], edge-cut width [29], and edge-treewidth [130].

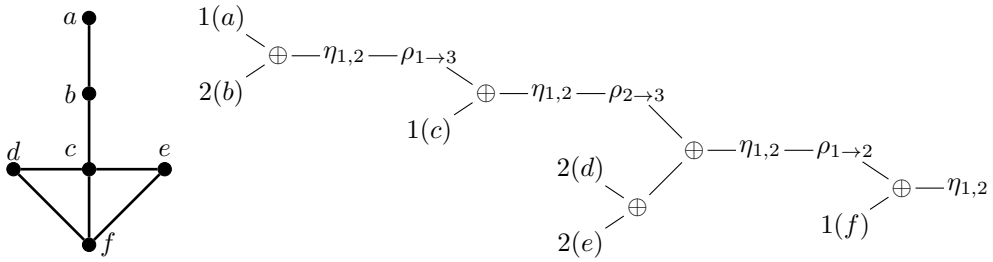


Fig. 2.3.: The right side depicts the associated syntax tree T_μ of a 3-expression μ that constructs the graph G on the left side, i.e., $G = G_\mu$.

Clique-Expressions and Clique-Width. A *labeled graph* is a graph $G = (V, E)$ together with a *label function* $\texttt{lab}\colon V \to \mathbb{N} = \{1, 2, 3, \ldots\}$; we usually omit mentioning $\texttt{lab}$ explicitly. A labeled graph is k-*labeled* if $\texttt{lab}(v) \leq k$ for all $v \in V$. We consider the following four operations on labeled graphs:

- the *introduce*-operation $\ell(v)$ which constructs a single-vertex graph whose unique vertex v has label ℓ,
- the *union*-operation $G_1 \oplus G_2$ which constructs the disjoint union of two labeled graphs G_1 and G_2,
- the *relabel*-operation $\rho_{i\to j}(G)$ changes the label of all vertices in G with label i to label j,
- the *join*-operation $\eta_{i,j}(G)$, $i \neq j$, which adds an edge between every vertex in G with label i and every vertex in G with label j.

A valid expression that only consists of introduce-, union-, relabel-, and join-operations is called a *clique-expression*. The graph constructed by a clique-expression μ is denoted G_μ and the constructed label function is denoted $\texttt{lab}_\mu\colon V(G_\mu) \to \mathbb{N}$.

We associate with a clique-expression μ the syntax tree T_μ in the natural way and we associate with each node $t \in V(T_\mu)$ the corresponding operation, see Figure 2.3. For any node $t \in V(T_\mu)$ the subtree rooted at t induces a *subexpression* μ_t.

We say that a clique-expression μ is a k-*clique-expression* or just k-*expression* if G_t is k-labeled for all $t \in V(T_\mu)$. The *clique-width* of a graph G, denoted by $\mathrm{cw}(G)$, is the minimum k such that there exists a k-expression μ with $G = G_\mu$. A clique-expression μ is *linear* if in every union-operation the second graph consists only of a single vertex. Accordingly, we also define the *linear-clique-width* of a graph G, denoted $\text{lin-cw}(G)$, by only considering linear clique-expressions.

A k-clique-expression, with k small, shows how a graph can be decomposed via *possibly large, but structured edge separators*. To be more precise, the edge separators induced by a k-clique-expression have small *neighborhood diversity*, i.e., the vertices on one side of the edge separator only have k possible types of neighborhoods on the other side of the separator. Ducoffe [62] formalizes this behavior using the notion of *partition trees*, which were introduced by Courcelle et al. [42], instead of clique-expressions.

Even approximating clique-width with sublinear additive error is NP-hard by Fellows et al. [69]. It is open whether clique-width can be computed by an XP algorithm and it

is even open if graphs of clique-width at most 4 can be recognized in polynomial time. A $(2^{2k+1}-1)$-approximation for clique-width can be computed in time $2^{2^{\mathcal{O}(k)}}n^2$ by Fomin and Korhonen [74] or a (2^{3k+1})-approximation in time $8^k n^4$ by Oum [147]; both algorithms actually approximate rankwidth, whose formal definition we omit here, which approximates clique-width via Lemma 2.4.3, thus yielding an exponential error.

Lemma 2.4.3 ([149]). *For any graph G, we have constructively[2] that $\mathrm{rw}(G) \leq \mathrm{cw}(G) \leq 2^{\mathrm{rw}(G)+1}-1$, where $\mathrm{rw}(G)$ is the rankwidth of G.*

Clique-width is more expressive than treewidth in the sense that every family of graphs with bounded treewidth also has bounded clique-width, but there are families with bounded clique-width and unbounded treewidth, such as the family of all cliques. However, Corneil and Rotics [40] show that the dependence between treewidth and clique-width can be exponential, cf. Lemma 2.4.4, which means in our setting that a single-exponential algorithm for some GRAPH PROBLEM[clique-width] a priori only results in a double-exponential algorithm, i.e., with running time $2^{2^{\mathcal{O}(k)}}n^{\mathcal{O}(1)}$, for GRAPH PROBLEM[treewidth].

Lemma 2.4.4 ([40]). *For any graph G, we have constructively that $\mathrm{cw}(G) \leq 3 \cdot 2^{\mathrm{tw}(G)-1}$. Furthermore, for every $k \in \mathbb{N}_{>0}$, there exists a graph G_k with $\mathrm{tw}(G) = k$ and $\mathrm{cw}(G) \geq 2^{\lfloor k/2 \rfloor - 1}$.*

The second part of Lemma 2.4.4 combined with the inequality $\mathrm{rw}(G) \leq \mathrm{tw}(G)+1$, which was proven by Oum [148], also implies that the gap between rankwidth and clique-width can be exponential. Hence, the exponential error when approximating clique-width via Lemma 2.4.3 cannot be avoided.

Multi-Clique-Width. Multi-clique-width is an extension of clique-width that has been introduced by Fürer [79] to bridge the exponential gap, cf. Lemma 2.4.4, between treewidth and clique-width. A *set-labeled graph* is a graph $G = (V, E)$ with a *set-label function* $\texttt{slab}\colon V \to \mathcal{P}(\mathbb{N})$, note that $\texttt{slab}(V) = \emptyset$ is allowed. A set-labeled graph is k-set-labeled if $\max(\texttt{slab}(v)) \leq k$ for all $v \in V$. Any label function naturally gives rise to a set-label function by replacing each label with the corresponding singleton set. A *multi-clique-expression* is a valid expression formed by the defining operations of clique-expressions (which we naturally extend to set-labeled graphs) and additionally the following two operations:

- the *set-relabel*-operation $\rho_{\ell \to S}$, $\emptyset \neq S \subseteq \mathbb{N}$, replaces label ℓ by the set of labels S, so that any label set S' with $\ell \in S'$ changes to $(S' \setminus \{\ell\}) \cup S$,
- the *deletion*-operation ϵ_ℓ deletes the label ℓ from all vertices.

All other terms are defined in analogy to clique-expressions. In particular, a *multi-k-expression* is a multi-clique-expression so that every subexpression yields a k-set-labeled graph. The *multi-clique-width* of a graph G, denoted $\mathrm{mcw}(G)$, is the minimum k such that there exists a multi-k-expression μ with $G = G_\mu$. We remark that the definition of Fürer [79] also allows for a more powerful introduce-operation that can introduce several vertices with several labels at once, but this can be simulated by multiple introduces, followed by multiple unions, and a set-relabel-operation without changing the multi-clique-width for any graph that contains at least one edge.

[2] Meaning that both inequalities come with polynomial-time algorithms appropriately transforming the decomposition for one parameter into a decomposition for the other parameter, see Section 2.4.4 for more details.

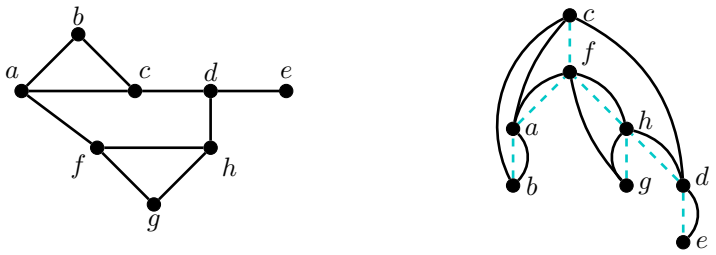


Fig. 2.4.: The right side depicts a treedepth decomposition $\mathcal{T} = (V, E_{\mathcal{T}})$ of depth 5 of the graph G on the left side. The dashed cyan edges denote the edges in $E_{\mathcal{T}}$.

Lemma 2.4.5 ([79]). *For every graph G, it holds constructively that* $\mathrm{mcw} \leq \mathrm{cw} \leq 2^{\mathrm{mcw}}$.

Very little is known about computing or approximating multi-clique-width. It is currently not known whether computing multi-clique-width is NP-hard. The only algorithms we are aware of are the algorithms for approximating clique-width via rank-width, which therefore also approximate multi-clique-width leading to another exponential increase of the error due to Lemma 2.4.5 and thus we only get a double-exponential error in total.

Dynamic Programming Conventions. We will see many dynamic programming algorithms on various graph decompositions in this thesis. Therefore, we have some conventions for the notation used to present them. Typically, each node t of a decomposition has some associated table A_t with an underlying family of partial solutions often denoted by $\mathcal{A}_t$. The table A_t is indexed by various objects: *target numbers* describing the size, cost, weight, etc. of the considered partial solutions and a *signature* f describing the *state* of the considered partial solutions. The target numbers are denoted by lowercase letters with a line over them, such as $\overline{c}$ or $\overline{w}$, and are written as superscripts of the table, i.e., $\mathcal{A}_t^{\overline{c},\overline{w}}$. A signature f is a function that maps a set associated with the node t to a problem-dependent set $\mathbf{States}$. The set $\mathbf{States}$ consists of formal symbols describing the possible states of a decomposition-dependent object, such as a vertex or a label class; the states are usually denoted by bold numbers or capital letters, such as $\mathbf{0}$, $\mathbf{1}$, or $\mathbf{A}$.

2.4.2 Beyond Width Parameters

We proceed by defining the relevant graph parameters that cannot be classified as width parameters.

Treedepth. Treedepth is the depth parameter equivalent of treewidth. A *treedepth decomposition* of an undirected graph $G = (V, E)$ is a rooted forest $\mathcal{T} = (V, E_{\mathcal{T}})$ such that for every edge $\{u, v\} \in E$ either u is an ancestor or descendant of v in $\mathcal{T}$. The *depth* of a rooted forest is the largest number of nodes on a path from a root to a leaf. The *treedepth* of G, denoted $\mathrm{td}(G)$, is the minimum depth over all treedepth decomposition of G.

See Figure 2.4 for an example of a treedepth decomposition. Treedepth decompositions are often also called *elimination forests/trees*. Indeed, treedepth has been studied under many different names such as *minimum elimination tree height* [21], *ordered chromatic*

number [113], and *vertex ranking* [20], and treedepth plays an important role in the theory of sparse graphs [143, 142, 141]. Formally supporting that the underlying decomposition scheme for width parameters and depth parameters is the same, Fürer and Yu [80] prove the following lemma; further parameter relationships will be discussed later.

Lemma 2.4.6 ([80])**.** *The treedepth of a graph $G = (V, E)$ is equal to the minimum forget-depth of (very) nice tree decompositions of G, where the forget-depth of a nice tree decomposition is the maximum number of forget bags on a root-leaf-path in the decomposition.*

Treedepth is NP-hard to compute even for bipartite graphs as shown by Bodlaender et al. [20]. Nadara et al. [135] show that treedepth can be computed exactly in time $2^{\mathcal{O}(k^2)}n$ and polynomial space and Czerwiński et al. [52] give an $\mathcal{O}(\mathrm{tw}\log^{3/2}\mathrm{tw})$-approximation in polynomial time.

Modulators. Given a family of graphs $\mathcal{F}$, an *$\mathcal{F}$-modulator* of a graph $G = (V, E)$ is a vertex set $X \subseteq V$ such that $G - X \in \mathcal{F}$. For a graph parameter p and constant $c \in \mathbb{N}$, define the family $\mathcal{F}_{\mathrm{p}\leq c} = \{G : \mathrm{p}(G) \leq c\}$ of graphs with parameter value at most c; we will use this notation for the parameters treewidth, pathwidth, and treedepth. We also call an $\mathcal{F}_{\mathrm{tw}\leq c}$-modulator a *modulator to treewidth c* and an $\mathcal{F}_{\mathrm{pw}\leq c}$-modulator a *modulator to pathwidth c*. We define $\mathrm{dist}_{\mathcal{F}}(G)$ as the minimum size of an $\mathcal{F}$-modulator of G; if $\mathcal{F} = \mathcal{F}_{\mathrm{p}\leq c}$ for some graph parameter p and constant c, we simply write $\mathrm{dist}_{\mathrm{p}\leq c}(G)$. Let $\mathcal{F}_{ind}$ be the family of all independent sets and $\mathcal{F}_{forest}$ be the family of all forests, we remark that a *vertex cover* is an $\mathcal{F}_{ind}$-modulator and a *feedback vertex set* is an $\mathcal{F}_{forest}$-modulator; we write $\mathrm{vc}(G) = \mathrm{dist}_{\mathcal{F}_{ind}}(G)$ and $\mathrm{fvs}(G) = \mathrm{dist}_{\mathcal{F}_{forest}}(G)$.

Lewis and Yannakakis [124] have shown that computing $\mathcal{F}$-modulators is NP-hard, whenever $\mathcal{F}$ is a nontrivial[3] family of graphs that is closed under taking subgraphs, which in particular applies to the cases considered by us. Fomin et al. [75] obtain general results on computing $\mathcal{F}$-modulators for minor-closed families $\mathcal{F}$ that do not contain all planar graphs, which includes $\mathcal{F}_{\mathrm{tw}\leq c}$ and $\mathcal{F}_{\mathrm{pw}\leq c}$; in particular, they present constant-factor approximation algorithms and single-exponential parameterized algorithms for such families.

2.4.3 Lifting to Twinclasses and Modules

So far we have defined a variety of graph parameters, but most of them only cover sparse graphs. We study two generic ways how a sparse graph parameter can be lifted to a graph parameter capable of capturing dense graphs. Both ways rely on partitioning the vertex set so that all vertices in one part of the partition have the same neighborhood in some sense. Given such a partition, we can contract each part of the partition into a single vertex and obtain a *quotient graph*, describing the adjacencies among the parts of the partition, on which we can measure the given parameter.

Lifting to Twinclasses

Quotients and Twins. Let Π be a partition of $V(G)$. The *quotient graph* G/Π is given by $V(G/\Pi) = \Pi$ and $E(G/\Pi) = \{\{B_1, B_2\} \subseteq \Pi : B_1 \neq B_2, \exists u \in B_1, v \in B_2 \colon \{u, v\} \in E(G)\}$.

[3]Nontrivial in this context means that the family has infinite size and avoids infinitely many graphs.

We say that two vertices u, v are *twins* if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. The equivalence classes of this relation are called *twinclasses* and we let $\Pi_{tc}(G)$ denote the partition of $V(G)$ into twinclasses. If $N(u) = N(v)$, then u and v are *false twins* and if $N[u] = N[v]$, then u and v are *true twins*. Every twinclass of size at least 2 consists of only false twins or only true twins. A false twinclass induces an independent set and a true twinclass induces a clique.

Lifting Graph Parameters. Given a graph parameter p, we define its *twinclass-variant* $\mathrm{tc\text{-}p}$ by setting $\mathrm{tc\text{-}p}(G) = \mathrm{p}(G/\Pi_{tc}(G))$ for all graphs G. In particular, we obtain the parameters *twinclass-treewidth*, *twinclass-pathwidth*, *twinclass-cutwidth*, and *twinclass-treedepth*. The parameters twinclass-treewidth and twinclass-pathwidth have been considered before under the name modular treewidth and modular pathwidth [123, 133, 150]. We use the prefix *twinclass* instead of *modular* to distinguish from the quotient graph arising from a *modular partition* of G. If we denote the number of vertices of a graph G by $n(G)$, then the parameter $\mathrm{tc\text{-}n}$ is also called *neighborhood diversity*, which was introduced by Lampis [122] and investigated further by Ganian [85] in the context of parameterized algorithms. We will see in Lemma 2.4.11 that the twinclass-variants of our considered parameters are NP-hard to compute, however algorithms for computing a decomposition/expression transfer as we can compute $G/\Pi_{tc}(G)$ in polynomial time.

We also lift $\mathcal{F}$-modulators into the twinclass setting. However, in this case, we do not simply measure $\mathrm{dist}_{\mathcal{F}}(G/\Pi_{tc}(G))$, but only apply the twinclass-lifting to the modular and not the remaining graph, leading to the following definition. Given a family of graphs $\mathcal{F}$, an *$\mathcal{F}$-twinclass-modulator (TCM)* $\mathcal{M} \subseteq \Pi_{tc}(G)$ of G is a set of twinclasses of G such that $G - \bigcup(\mathcal{M}) \in \mathcal{F}$. The *size* of a twinclass-modulator $\mathcal{M}$ is $|\mathcal{M}|$, i.e., the number of twinclasses $\mathcal{M}$ contains. We define $\mathrm{tcm}_{\mathcal{F}}(G)$ as the minimum size of an $\mathcal{F}$-twinclass-modulator of G. For families of the form $\mathcal{F}_{\mathrm{p}\leq c}$, we introduce similar abbreviations as for the standard modulators.

Lifting to Modules

Similar to twinclass-lifting, we can also use the *modular decomposition*, a well-known tool to capture dense graphs, of a graph to turn sparse graph parameters into dense parameters. Notable differences to the twinclass-case are that, in general, a modular decomposition yields several quotient graphs and not only a single one. Additionally, some quotient graphs can be considered edge cases on which we will not measure the given parameter. However, we gain that the modular-variant of a parameter is more expressive than the twinclass-variant.

Modular Decomposition. A vertex set $M \subseteq V(G)$ is a *module* of G if $N(v) \setminus M = N(w) \setminus M$ for every pair $v, w \in M$ of vertices in M. Equivalently, for every $u \in V(G) \setminus M$ it holds that $M \subseteq N(u)$ or $M \cap N(u) = \emptyset$. In particular, every twinclass is a module. We let $\mathcal{M}(G)$ denote the set of all modules of G. The modules $\emptyset$, $V(G)$, and all singletons are called *trivial*. A *prime graph* is a graph that only admits trivial modules. If $M \neq V(G)$, then we say that M is *proper*. For two disjoint modules $M_1, M_2 \in \mathcal{M}(G)$, either $\{\{v, w\} : v \in M_1, w \in M_2\} \subseteq E(G)$ or $\{\{v, w\} : v \in M_1, w \in M_2\} \cap E(G) = \emptyset$; in the first case, M_1 and M_2 are *adjacent* and in the second case, they are *nonadjacent*.

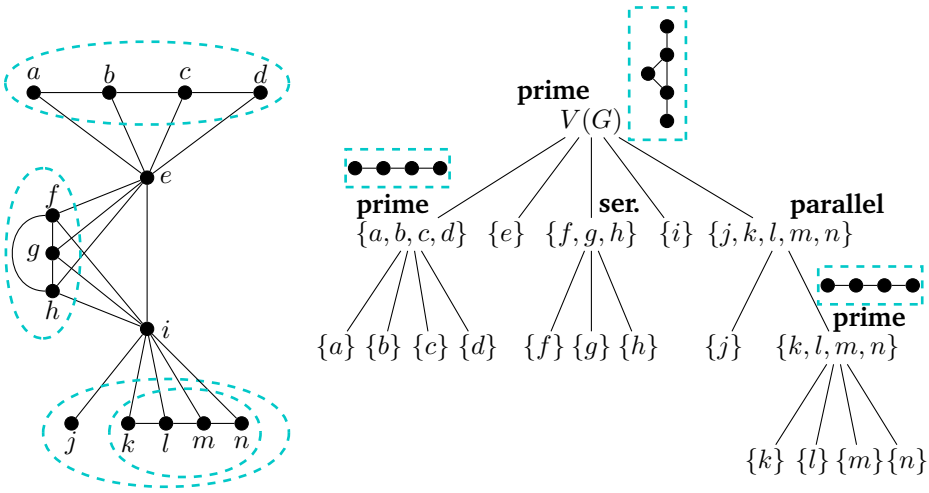


Fig. 2.5.: The left side depicts a graph G, where all modules in $\mathcal{M}^*_{\text{tree}}(G) \setminus \{V(G)\}$ are highlighted by dashed cyan ellipses. The right side depicts the modular decomposition tree of G, where all internal nodes are annotated with their type according to Theorem 2.4.7; in the case of prime nodes, also the corresponding quotient graph is shown in a dashed cyan rectangle.

A module M is *strong* if for every module $M' \in \mathcal{M}(G)$ we have that $M \cap M' = \emptyset$, $M \subseteq M'$, or $M' \subseteq M$, so strong modules intersect other modules only in a trivial way. Let $\mathcal{M}_s(G)$ denote the set of all strong modules of G. The defining property of strong modules implies that $\mathcal{M}_s(G)$ is a *laminar set family*[4]. Thus, the Hasse diagram[5] of $\mathcal{M}_{\text{tree}}(G) = \mathcal{M}_s(G) \setminus \{\emptyset\}$ with the inclusion-relation, is a rooted tree, called the *modular decomposition (tree)* of G. We freely switch between viewing $\mathcal{M}_{\text{tree}}(G)$ as a set family or as the modular decomposition tree of G. In the latter case, we usually speak of *nodes* of the modular decomposition tree.

Every graph G with at least two vertices can be uniquely partitioned into a set of inclusion-maximal non-trivial strong modules $\Pi_{mod}(G) = \{M_1, \ldots, M_\ell\}$, with $\ell \geq 2$, called *canonical modular partition*. For $M \in \mathcal{M}_{\text{tree}}(G)$ with $|M| \geq 2$, let $\texttt{children}(M) = \Pi_{mod}(G[M])$ as the sets in $\Pi_{mod}(G[M])$ are precisely the children of M in the modular decomposition tree; if $|M| = 1$, then $\texttt{children}(M) = \emptyset$. We write $\mathcal{M}^*_{\text{tree}}(G) = \mathcal{M}_{\text{tree}}(G) \setminus \{\{v\} : v \in V\}$. Each internal node $M \in \mathcal{M}^*_{\text{tree}}(G)$ of the modular decomposition tree is associated with a *quotient graph* $G^q_M = G[M]/\Pi_{mod}(G[M])$, which captures the adjacencies among the child modules of M. The original graph G can be reconstructed from its modular decomposition tree and all associated quotient graphs. The quotient graphs occurring in the modular decomposition tree can be classified into three cases:

Theorem 2.4.7 ([84]). *For $M \in \mathcal{M}^*_{\text{tree}}(G)$, exactly one of the following holds:*

- ***Parallel node**: $G[M]$ is not connected and G^q_M is an independent set,*
- ***Series node**: the complement $\overline{G[M]}$ is not connected and G^q_M is a clique,*
- ***Prime node**: $\Pi_{mod}(G[M])$ consists of the inclusion-maximal proper modules of $G[M]$ and G^q_M is a prime graph, i.e., G^q_M has no nontrivial modules.*

[4] A set family $\mathcal{F}$ is *laminar* if for every $S_1, S_2 \in \mathcal{F}$ it holds that $S_1 \subseteq S_2$, $S_2 \subseteq S_1$, or $S_1 \cap S_2 = \emptyset$

[5] The Hasse diagram is a drawing of the transitive reduction, i.e., there is an edge from $M_1 \in \mathcal{M}_{\text{tree}}(G)$ to $M_2 \in \mathcal{M}_{\text{tree}}(G)$ if $M_1 \subsetneq M_2$ and there is no $M_3 \in \mathcal{M}_{\text{tree}}(G)$ with $M_1 \subsetneq M_3 \subsetneq M_2$.

See Figure 2.5 for an example of a modular decomposition tree. We emphasize that when we say that a module M is a prime node of G, it means that the associated quotient graph G^q_M is a prime graph, whereas the graph $G[M]$ is not necessarily a prime graph. We collect the graphs that appear as prime quotient graphs in the modular decomposition of G in the family $\mathcal{H}_p(G) = \{G^q_M : M \in \mathcal{M}^*_{\text{tree}}(G), G^q_M \text{ is prime}\}$. The modular decomposition tree can be computed in time $\mathcal{O}(n+m)$, see e.g. Tedder et al. [169] or the survey by Habib and Paul [97].

Let $M \in \mathcal{M}_{\text{tree}}(G) \setminus \{V\}$ and $M^\uparrow \in \mathcal{M}_{\text{tree}}(G)$ be its *parent module*. We have that $M \in \Pi_{mod}(G[M^\uparrow])$, hence M appears as a vertex of the quotient graph $G^q_{M^\uparrow}$; we will also denote this vertex by v^q_M. Note that $G^q_{M^\uparrow}$ is the only quotient graph in the modular decomposition of G where M appears as a vertex. So, we implicitly know that $v^q_M \in V(G^q_{M^\uparrow})$ without having to specify $M^\uparrow$.

Lifting Graph Parameters. Many graph problems can be solved by working only on $\mathcal{H}_p(G)$. Hence, we consider the values of standard graph parameters on $\mathcal{H}_p(G)$. Given a graph parameter p, we define its *modular-variant* by setting $\text{mod-p}(G) = \max(1, \max_{H \in \mathcal{H}_p(G)} \mathrm{p}(H))$ for all graphs G. In particular, we obtain the parameters *modular-treewidth*, *modular-pathwidth*, *modular-cutwidth*, and *modular-treedepth*. This version of modular-treewidth was first used by Bodlaender and Jansen for Maximum Cut [22]. Kratsch and Nelles [120] combine modular decompositions with tree-depth in various ways and obtain parameterized algorithms for various efficiently solvable problems. If we denote the number of vertices of a graph G by $n(G)$, then the parameter mod-n is essentially the parameter *modular-width*, for which Gajarsky et al. [83] and Kratsch and Nelles [119] obtain several parameterized algorithms. Ganian et al. [90] study so-called *rank-width-d covers*, which are very similar to the parameter modular-rank-width, in the context of meta-kernelization.

Computing Modular-Variants. Lemma 2.4.11 will show that the modular-variants of the considered parameters are NP-hard to compute. As the modular decomposition tree can be computed in time $\mathcal{O}(n+m)$ via, e.g., the algorithm of Tedder et al. [169], we can run algorithms computing the original graph parameter on all the resulting quotient graphs and thus obtain an algorithm for computing the modular-variant. As a consequence, we obtain reasonable algorithms for approximating or exactly computing modular-variants of several graph parameters, which we are still lacking for (multi-)clique-width. The following theorem illustrates this approach in more detail for modular-treewidth.

Theorem 2.4.8. *If $\mathbb{A}_{\text{tw}}$ is an algorithm that given an n-vertex graph G and an integer k, in time $\mathcal{O}(f(k)n^c)$, $c \geq 1$, either outputs a tree decomposition of width at most $g(k)$ or determines that $\text{tw}(G) > k$, then there is an algorithm $\mathbb{A}_{\text{mod-tw}}$ that given an n-vertex m-edge graph G and an integer k, in time $\mathcal{O}(f(k)n^c + m)$ either outputs a tree decomposition of width at most $g(k)$ for every prime quotient graph $G^q_M \in \mathcal{H}_p(G)$ or determines that $\text{mod-tw}(G) > k$.*

Proof. The algorithm $\mathbb{A}_{\text{mod-tw}}$ works as follows. We first compute the modular decomposition tree of G in time $\mathcal{O}(n+m)$ with, e.g., the algorithm of Tedder et al. [169] and obtain the family of prime quotient graphs $\mathcal{H}_p(G)$. Then, we run $\mathbb{A}_{\text{tw}}$ on every $H \in \mathcal{H}_p(G)$; reporting that $\text{mod-tw}(G) > k$ if $\mathbb{A}_{\text{tw}}$ reports $\text{tw}(H)$ for some $H \in \mathcal{H}_p(G)$. Since the modular

decomposition tree has n leaves and every internal node has at least two children, we obtain that $|\mathcal{M}_{\text{tree}}(G)| \leq 2n$. This also implies that $\sum_{H \in \mathcal{H}_p(G)} |V(H)| \leq 2n$, since the vertices of the quotient graph G^q_M at $M \in \mathcal{M}^*_{\text{tree}}(G)$ are in one-to-one correspondence with the children of M in the modular decomposition tree. Neglecting the constant term, we can bound the running time of all $\mathbb{A}_{\text{tw}}$ by using a calculation similar to Kratsch and Nelles [119] as follows:

$$\sum_{H \in \mathcal{H}_p(G)} f(k)|V(H)|^c \leq f(k)n^{c-1} \sum_{H \in \mathcal{H}_p(G)} |V(H)| \leq 2f(k)|V(H)|^c.$$

The algorithm is clearly correct, so this concludes the proof. □

Corollary 2.4.9. *There is an algorithm, that given an n-vertex graph G and an integer k, in time $2^{\mathcal{O}(k)}n + m$ either outputs a tree decomposition of width at most $2k+1$ for every prime quotient graph $G^q_M \in \mathcal{H}_p(G)$ or determines that $\text{mod-tw}(G) > k$.*

Proof. We apply Theorem 2.4.8 with the algorithm of Korhonen [117] that satisfies $f(k) = 2^{\mathcal{O}(k)}$ and $g(k) = 2k+1$. □

Properties of Twinclass- and Modular-Variants

The following theorem implies that if a graph parameter p is monotone wrt. induced subgraphs, then also tc-p and mod-p are monotone wrt. induced subgraphs.

Lemma 2.4.10. *Let G, H be graphs. If H is an induced subgraph of G, then $H^q_{tc} = H/\Pi_{tc}(H)$ is an induced subgraph of $G^q_{tc} = G/\Pi_{tc}(G)$ and for every graph $\widetilde{H} \in \mathcal{H}_p(H)$ there is some graph $\widetilde{G} \in \mathcal{H}_p(G)$ such that $\widetilde{H}$ is an induced subgraph of $\widetilde{G}$.*

Proof. For the first part, note that if two vertices $v, w \in V(H) \subseteq V(G)$ are twins in G, then they are also twins in H. Therefore, every twinclass of G intersects at most one twinclass of H. Consider some map $\iota \colon V(H^q_{tc}) = \Pi_{tc}(H) \to V(G^q_{tc}) = \Pi_{tc}(G)$ that maps each twinclass C of H to some twinclass of G intersecting C; the precise choice of the twinclass of G does not matter. The map ι is injective by the previous argument. We have $\{C, C'\} \in E(H^q_{tc})$ if and only if $\{\iota(C), \iota(C')\} \in E(G^q_{tc})$, therefore H^q_{tc} is an induced subgraph of G^q_{tc}.

The second part is more involved. Consider some $\widetilde{H} \in \mathcal{H}_p(H)$ and let $\widetilde{M}_H \in \mathcal{M}^*_{\text{tree}}(H)$ such that $\widetilde{H} = H[\widetilde{M}_H]/\Pi_{mod}(H[\widetilde{M}_H])$. Pick the inclusion-minimal $\widetilde{M}_G \in \mathcal{M}^*_{\text{tree}}(G)$ such that $\widetilde{M}_H \subseteq \widetilde{M}_G$; $\widetilde{M}_G$ is unique since $\mathcal{M}^*_{\text{tree}}(G)$ is a laminar set family. We first claim that $\widetilde{M}_G$ is a prime node of G. By Theorem 2.4.7, it suffices to show that $G[\widetilde{M}_G]$ and its complement graph are connected. Since $\widetilde{M}_H$ is a prime node of H, we already know that $H[\widetilde{M}_H]$ and its complement graph are connected. Therefore, $H[\widetilde{M}_H] = G[\widetilde{M}_H]$ is part of a connected component of $G[\widetilde{M}_G]$. If $G[\widetilde{M}_G]$ were disconnected, then $\widetilde{M}_G$ is not minimal, since the connected component of $G[\widetilde{M}_G]$ containing $\widetilde{M}_H$ would induce a smaller module. Since also the complement graph of $H[\widetilde{M}_H]$ must be an induced subgraph of the complement graph of $G[\widetilde{M}_G]$, we can repeat the same argument to see that also the complement graph of $G[\widetilde{M}_G]$ is connected. Therefore, $\widetilde{M}_G$ is a prime node of G.

Now, set $\widetilde{G} = G[\widetilde{M}_G]/\Pi_{mod}(G[\widetilde{M}_G]) \in \mathcal{H}_p(G)$. We claim that $\widetilde{H}$ is an induced subgraph of $\widetilde{G}$. Consider any $M_G \in \Pi_{mod}(G[\widetilde{M}_G])$ with $M_G \cap \widetilde{M}_H \neq \emptyset$. We claim that $M_G \cap \widetilde{M}_H$ is a (not necessarily strong) module of $H[\widetilde{M}_H]$: pick any $v, w \in M_G \cap \widetilde{M}_H$ and compute

$$\begin{aligned} N_{H[\widetilde{M}_H]}(v) \setminus M_G &= (N_G(v) \cap \widetilde{M}_H) \setminus M_G = (N_G(w) \cap \widetilde{M}_H) \setminus M_G \\ &= N_{H[\widetilde{M}_H]}(w) \setminus M_G, \end{aligned}$$

where the second equality uses that M_G is a module of G. We cannot have $\widetilde{M}_H \subseteq M_G$ by minimality of $\widetilde{M}_G$, hence $M_G \cap \widetilde{M}_H$ is a proper module of $H[\widetilde{M}_H]$. By Theorem 2.4.7, $\Pi_{mod}(H[\widetilde{M}_H])$ consists of all inclusion-maximal proper modules of $H[\widetilde{M}_H]$ and therefore M_G can intersect at most one module in $\Pi_{mod}(H[\widetilde{M}_H])$, otherwise the maximality would be violated for some modules. Similarly to the twinclass case, we can now define an injective map $\iota \colon \Pi_{mod}(H[\widetilde{M}_H]) \to \Pi_{mod}(G[\widetilde{M}_G])$ mapping each module $M_H \in \Pi_{mod}(H[\widetilde{M}_H])$ to some module $M_G \in \Pi_{mod}(G[\widetilde{M}_G])$ intersecting M_H and witnessing that $\widetilde{H}$ is an induced subgraph of $\widetilde{G}$. □

By giving an essentially parameter-preserving graph transformation so that all nontrivial twinclasses and modules are destroyed, we can show that many twinclass- and modular-variants are NP-hard to compute.

Lemma 2.4.11. *The parameters* tc-td, tc-ctw, tc-pw, tc-tw, mod-td, mod-ctw, mod-pw *and* mod-tw *are all NP-hard to compute.*

Proof. We show all of these hardness results by the same approach: we transform a given graph G into a graph G' such that G' has a trivial modular structure, i.e., G' is a prime graph, and such that $\mathrm{p}(G) = \mathrm{p}(G') + \mathcal{O}(1)$ for $\mathrm{p} \in \{\mathrm{td}, \mathrm{ctw}, \mathrm{pw}, \mathrm{tw}\}$. Since G' is prime, we then have that $\text{mod-p}(G') = \text{tc-p}(G') = \mathrm{p}(G') = \mathrm{p}(G) + \mathcal{O}(1)$ and we have reduced the computation of p to its twinclass-variant tc-p or modular-variant mod-p, thus the NP-hardness transfers.

We begin by handling the parameters treedepth, pathwidth, and treewidth. Without loss of generality, assume that $G = (V, E)$ is a connected graph with at least two vertices. The graph $G' = (V \cup V', E \cup E')$, with $V' = \{v' : v \in V\}$ and $E' = \{\{v, v'\} : v \in V\}$, is obtained from G by attaching a private degree-1 vertex to each vertex of G.

We claim that G' is a prime graph, i.e., G' only admits trivial modules and hence also only trivial twinclasses. Suppose that $\emptyset \neq M \neq V(G')$ is a module of G' with $|M| \geq 2$. If $v \in M$, then also $v' \in M$, as otherwise $N_{G'}(v) \setminus M \neq N_{G'}(w) \setminus M$ for every $w \in M \setminus \{v\} \neq \emptyset$. If $|M \cap V'| \leq 1$, then we must have $M = \{v, v'\}$ for some $v \in V$ by the previous argument, but this cannot be a module of G', since there is at least one vertex $w \in V$ that is adjacent to v but not to v' by assumption on G. If $|M \cap V'| \geq 2$, then $v' \in M$ also implies $v \in M$, as v is adjacent to v' but not to $w' \in (M \cap V') \setminus \{v'\} \neq \emptyset$. Hence, we have $\{v, v'\} \subseteq M$ or $\{v, v'\} \cap M = \emptyset$ for every $v \in V$. Since $M \neq V(G')$ and G is connected, there exists some $v \in M \cap V$ and $w \in N_{G'}(v) \setminus M$, but w is not adjacent to v', so M cannot be a module. This proves the claim.

We have that $\mathrm{tw}(G') = \mathrm{tw}(G)$, since G is a subgraph of G' and every tree decomposition of G gives rise to a tree decomposition of G' of the same width by picking for every vertex $v \in V$ some bag B with $v \in B$ and attaching the bag $\{v, v'\}$ as a degree-1 node. Hence,

$\mathrm{tc\text{-}tw}$ and $\mathrm{mod\text{-}tw}$ must be NP-hard to compute, since Arnborg et al. [4] have shown that treewidth is NP-hard to compute.

We claim that $\mathrm{td}(G') = \mathrm{td}(G) + 1$ and hence $\mathrm{tc\text{-}td}$ and $\mathrm{mod\text{-}td}$ must be NP-hard to compute, since Pothen [157] has shown that treedepth is NP-hard to compute. We have $\mathrm{td}(G') \leq \mathrm{td}(G) + 1$, since a treedepth decomposition for G' can be obtained by taking a treedepth decomposition of G and attaching v' as a child of v for every $v \in V$. We have $\mathrm{td}(G') \geq \mathrm{td}(G) + 1$, since every root-leaf path in a treedepth decomposition of G' must contain at least one vertex of V'; otherwise some $v \in V$ is a leaf in the treedepth decomposition of G' for which v and v' cannot be in an ancestor-descendant-relationship.

For pathwidth, we only have that $\mathrm{pw}(G) \leq \mathrm{pw}(G') \leq \mathrm{pw}(G) + 1$, but this still allows us to conclude that $\mathrm{mod\text{-}pw}$ and $\mathrm{tc\text{-}pw}$ are NP-hard to compute as Bodlaender et al. [21] have shown that even approximating pathwidth up to any constant absolute error is NP-hard. It remains to prove the two inequalities. The first one $\mathrm{pw}(G) \leq \mathrm{pw}(G')$ immediately follows, since G is a subgraph of G'. For the second inequality $\mathrm{pw}(G') \leq \mathrm{pw}(G) + 1$, we consider any very nice path decomposition of G of width k and transform it into a path decomposition of G' of width at most $k + 1$ as follows. For every $v \in V$, we consider its introduce bag B^v and add the bag $B^{v'} := B^v \cup \{v'\}$ between B^v and its original successor, then all vertices and edges of G' are covered and the width of the decomposition only increases by 1.

It remains to handle the parameter cutwidth. Here, we choose a different construction for G'. We again assume that $G = (V, E)$ is connected and contains at least two vertices. We construct $G' = (V', E')$ by subdividing every edge of G twice, i.e., every edge $\{v, w\} \in E$ is replaced by two new vertices $s^v_{\{v,w\}}, s^w_{\{v,w\}}$ and the edges $\{v, s^v_{\{v,w\}}\}$, $\{s^v_{\{v,w\}}, s^w_{\{v,w\}}\}$, $\{s^w_{\{v,w\}}, w\}$. This construction was also used by Bojikian et al. [24] to solve several problems parameterized by cutwidth. We proceed by showing that G' is prime.

Suppose that M is a module of G' with $|M| \geq 2$. If M contains two original vertices $v, w \in V$, then M must contain all subdivision vertices adjacent to them, as each subdivision vertex is adjacent to exactly one original vertex. If $|M \cap \{v, s^v_{\{v,w\}}, s^w_{\{v,w\}}, w\}| \geq 2$ for some edge $\{v, w\} \in E$, then we must have $\{v, s^v_{\{v,w\}}, s^w_{\{v,w\}}, w\} \subseteq M$ as the induced path on four vertices is a prime graph. We cannot have $|M| \geq 2$ and $|M \cap \{v, s^v_{\{v,w\}}, s^w_{\{v,w\}}, w\}| = 1$ for some edge $\{v, w\} \in E$, as otherwise there exists a subdivision vertex outside of M that is adjacent to some vertices of M, but not to all of them. By using that G' is connected and applying these three arguments repeatedly, we see that $|M| \geq 2$ implies $M = V'$ and G' can therefore not have any nontrivial modules.

We claim that $\mathrm{ctw}(G) = \mathrm{ctw}(G')$ and hence $\mathrm{tc\text{-}ctw}$ and $\mathrm{mod\text{-}ctw}$ are NP-hard to compute, as Garey and Johnson [91] have shown that cutwidth is NP-hard to compute. To prove the claim, we argue that subdividing any edge $\{v, w\}$ of a graph does not change the cutwidth. Let s denote the resulting subdivision vertex. We can account for s by adding it anywhere between v and w in the original layout, which is easily seen to not increase the width of the layout. For the other direction, we can assume without loss of generality that s lies between v and w in the linear layout, then removing s yields a linear layout for G of the same width. □

2.4.4 Parameter Relationships

As we want to investigate how different input structure quantified via graph parameters impacts problem complexity, it is particularly important to understand how the various graph parameters are related to each other. Due to such relations, the obtained algorithms and lower bounds are not only isolated results, but rather paint a larger picture describing how the problem complexity changes when moving towards a more or less restrictive input structure.

The Relation $\preceq$. Let p, q be two graph parameters, we write $\mathrm{p} \preceq \mathrm{q}$ if there exists some $c \in \mathbb{N}_0$ such that $\mathrm{p}(G) \leq \mathrm{q}(G) + c$ for all graphs G. If $\mathrm{p} \preceq \mathrm{q}$, then p is at least as *expressive* as q and q is at least as *restrictive* as p. The relation $\preceq$ is a *preorder* as it is transitive and reflexive. However, it is not a partial order, as we lack antisymmetry, since $\mathrm{p} \preceq \mathrm{q}$ and $\mathrm{q} \preceq \mathrm{p}$ only implies that p and q differ by at most a constant on all graphs and not necessarily that $\mathrm{p} = \mathrm{q}$. If $\mathrm{p} \preceq \mathrm{q}$, then algorithms for parameterization by p essentially carry over to parameterization by q, and lower bounds carry over in the other direction.

The Relation $\preceq^*$. Since we only consider single-exponential algorithms in this thesis, a less restrictive inequality suffices for algorithms to carry over so that the exponential dependence remains the same, which will be formalized in the forthcoming lemma. Let p, q be two graph parameters, we write $\mathrm{p} \preceq^* \mathrm{q}$ if there exists some $c \in \mathbb{N}_0$ such that $\mathrm{p}(G) \leq \mathrm{q}(G) + c \log(|V(G)| + 1)$ for all graphs G, where the 1 in the logarithm handles the empty graph. Essentially, the logarithmic term transforms into a polynomial factor for single-exponential algorithms, cf. Lemma 2.4.12. Again, $\preceq^*$ is a preorder, but not a partial order, and $\mathrm{p} \preceq \mathrm{q}$ implies $\mathrm{p} \preceq^* \mathrm{q}$.

Lemma 2.4.12. *Let p, q be two graph parameters and $\alpha \in \mathbb{R}_{>1}$. If $\mathrm{p} \preceq^* \mathrm{q}$, then $\mathcal{O}^*(\alpha^{\mathrm{p}(G)}) \leq \mathcal{O}^*(\alpha^{\mathrm{q}(G)})$.*

Proof. Since there is some $c \in \mathbb{N}_0$ such that $\mathrm{p}(G) \leq \mathrm{q}(G) + c \log(|V(G)| + 1)$ for all graphs G, we compute for all graphs G that

$$\begin{aligned}\alpha^{\mathrm{p}(G)} &\leq \alpha^{\mathrm{q}(G) + c \log(|V(G)|+1)} = \alpha^{\mathrm{q}(G)} 2^{c \log(|V(G)|+1) \log \alpha} = \alpha^{\mathrm{q}(G)} (|V(G)|+1)^{c \log \alpha} \\ &\leq \alpha^{\mathrm{q}(G)} |V(G)|^{\mathcal{O}(1)}.\end{aligned}$$

□

Witnesses and Constructive Inequalities. Almost all considered graph parameters p in this thesis are known to be NP-hard to compute, but, in all considered cases, inequalities of the form $\mathrm{p}(G) \leq k$ can be verified in polynomial time given an appropriate witness, which is typically some graph decomposition or expression. We say that a witness for $\mathrm{p}(G) \leq k$ is a p-*decomposition of* G of value k.[6] We remark that for e.g. modulators to treewidth c, a decomposition consists of a modulator and an appropriate tree decomposition of the remaining graph. As all considered algorithms assume that a decomposition is already given, the relations $\mathrm{p} \preceq \mathrm{q}$ and $\mathrm{p} \preceq^* \mathrm{q}$ are only algorithmically useful if a q-decomposition of G can

[6] We only use this notion informally. Without explicitly fixing a verification algorithm, there can be many structurally different witnesses for "$\mathrm{p}(G) \leq k$", but we will only consider the "canonical witnesses/decompositions" given by the parameter definition.

be efficiently transformed into a p-decomposition of G with the appropriate value. We say that an inequality $\mathrm{p}(G) \leq f(\mathrm{q}(G))$, for some computable function f, between two graph parameters p and q is *constructive* if it comes with a polynomial-time algorithm transform a q-decomposition of value k into a p-decomposition of value $f(k)$.

Transfer of Algorithms and Lower Bounds. All considered inequalities between parameters will be constructive. Hence, if $\mathrm{p} \preceq^* \mathrm{q}$ and Graph Problem[p] is solvable in time $\mathcal{O}^*(\alpha^k)$, then Graph Problem[q] is also solvable in time $\mathcal{O}^*(\alpha^k)$, by first transforming the q-decomposition into an appropriate p-decomposition and then running the algorithm for Graph Problem[p], where we use Lemma 2.4.12 to transfer the running time. On the other hand, if $\mathrm{p} \preceq^* q$ and Graph Problem[q] is *not* solvable in time $\mathcal{O}^*(\alpha^k)$, then also Graph Problem[p] is not solvable in time $\mathcal{O}^*(\alpha^k)$.

Relationship Overview

Figure 2.6 shows an overview of the relations $\preceq$ and $\preceq^*$ for the graph parameters considered in this thesis. We will provide proofs or cite appropriate references for these relationships in the remaining part of this subsection.

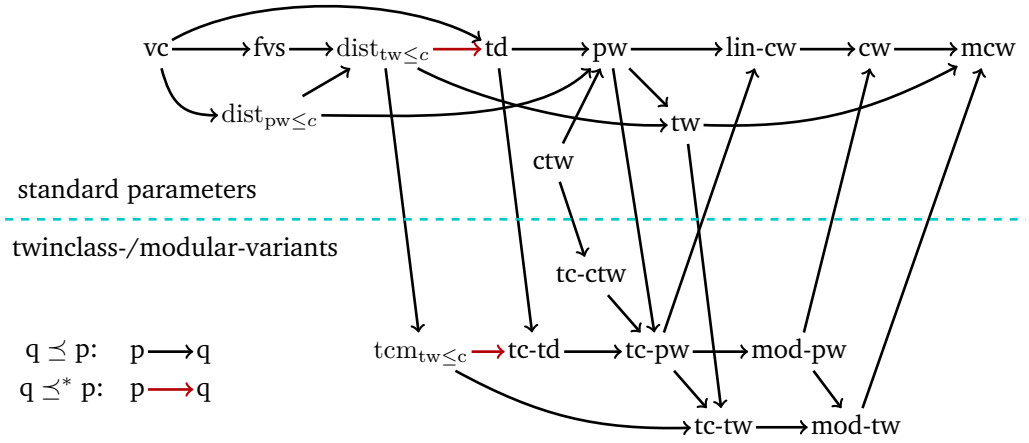


Fig. 2.6.: The proven or cited relationships between the graph parameters considered in this thesis. Due to transitivity, any path consisting of only black arrows from parameter p to q implies that $\mathrm{q} \preceq \mathrm{p}$. If the path contains at least one red arrow, then we can only conclude $\mathrm{q} \preceq^* \mathrm{p}$. Roughly, the parameters on the left are more restrictive than the parameters on the right.

Known Inequalities Among Standard Graph Parameters

We begin by listing the known relationships between standard graph parameters, i.e., as opposed to their twinclass- and modular-variants, considered in this thesis. These inequalities cover the relationships among the studied sparse parameters and some of the known relationships for clique-width.

Lemma 2.4.13. *(Folklore) We have constructively that* $\mathrm{fvs} \preceq \mathrm{vc}$ *and for every constant* $c \in \mathbb{N}_{>0}$ *it holds constructively that* $\mathrm{dist}_{\mathrm{pw}\leq c} \preceq \mathrm{vc}$, $\mathrm{dist}_{\mathrm{tw}\leq c} \preceq \mathrm{fvs}$, *and* $\mathrm{dist}_{\mathrm{tw}\leq c} \preceq \mathrm{dist}_{\mathrm{pw}\leq c}$.

Proof. All these inequalities follow from the fact that for two graph families $\mathcal{F}_1$ and $\mathcal{F}_2$ with $\mathcal{F}_1 \subseteq \mathcal{F}_2$, any $\mathcal{F}_1$-modulator is also an $\mathcal{F}_2$-modulator and therefore $\mathrm{dist}_{\mathcal{F}_2}(G) \leq \mathrm{dist}_{\mathcal{F}_1}(G)$ for all graphs G. □

Lemma 2.4.14. *(Folklore) For every constant $c \in \mathbb{N}_{>0}$ it holds constructively that* $\mathrm{td} \preceq \mathrm{dist}_{\mathrm{td}\leq c}$, $\mathrm{pw} \preceq \mathrm{dist}_{\mathrm{pw}\leq c}$, *and* $\mathrm{tw} \preceq \mathrm{dist}_{\mathrm{tw}\leq c}$. *In particular, we have constructively that* $\mathrm{td} \preceq \mathrm{vc}$.

Proof. If $\mathrm{p} \in \{\mathrm{td}, \mathrm{pw}, \mathrm{tw}\}$, then it holds for all graphs G that $\mathrm{p}(G) \leq \mathrm{dist}_{\mathrm{p}\leq c}(G) + c$. Given an $\mathcal{F}_{\mathrm{p}\leq c}$-modulator X, we add X to every bag of a path/tree decomposition for $G - X$ if $\mathrm{p} \in \{\mathrm{pw}, \mathrm{tw}\}$. For $\mathrm{p} = \mathrm{td}$, we create a treedepth decomposition for G by starting with a path corresponding to X and attaching a treedepth decomposition for $G - X$ below this path. The final inequality follows due to $\mathrm{vc} = \mathrm{dist}_{\mathrm{td}\leq 1}$. □

Lemma 2.4.15. *(Folklore) We have constructively that* $\mathrm{tw} \preceq \mathrm{pw}$ *and* $\mathrm{mcw} \preceq \mathrm{cw} \preceq \text{lin-cw}$.

Proof. All these inequalities follow since the parameters on the right are defined by the same decomposition as on the left, but with additional constraints. □

Lemma 2.4.16 ([18])**.** *For any graph G, it holds constructively that* $\mathrm{pw}(G) \leq \mathrm{ctw}(G)$, *hence* $\mathrm{pw} \preceq \mathrm{ctw}$.

Lemma 2.4.17 (Chapter 6 of [142])**.** *For any graph G, it holds constructively that* $\mathrm{pw}(G) \leq \mathrm{td}(G) - 1$, $\mathrm{td}(G) \leq (\mathrm{tw}(G) + 1)\log_2 |V(G)|$, *and* $\mathrm{td}(G) \leq \mathrm{td}(G - v) + 1$ *for any vertex* $v \in V(G)$.

Corollary 2.4.18. *For any constant c and graph G, it holds constructively that* $\mathrm{td}(G) \leq \mathrm{dist}_{\mathrm{tw}\leq c}(G) + (c+1)\log_2 |V(G)|$ *and therefore* $\mathrm{td} \preceq^* \mathrm{dist}_{\mathrm{tw}\leq c}$.

Proof. Let X be a modulator to treewidth c for G. Lemma 2.4.17 implies that $\mathrm{td}(G - X) \leq (c+1)\log_2 |V(G)|$. By repeatedly invoking the inequality $\mathrm{td}(G) \leq td(G - v) + 1$ of the same lemma for $v \in X$, we therefore obtain $\mathrm{td}(G) \leq |X| + (c+1)\log_2 |V(G)|$. □

Lemma 2.4.19 ([69])**.** *For any graph G, we have that* $\mathrm{cw}(G) \leq \text{lin-cw}(G) \leq \mathrm{pw}(G) + 2$ *constructively, hence* $\mathrm{cw} \preceq \text{lin-cw} \preceq \mathrm{pw}$.

Lemma 2.4.20 ([79])**.** *For every graph G, it holds that* $\mathrm{mcw}(G) \leq \mathrm{tw}(G) + 2$ *constructively and hence* $\mathrm{mcw} \preceq \mathrm{tw}$.

Relationships Involving Twinclass- or Modular-Variants

We now consider also relationships involving twinclass- or modular-variants of the considered parameters. We begin by proving a generic lemma implying that relationships among the original parameters carry over to their twinclass- or modular-variants.

Lemma 2.4.21. *Let p, q be two graph parameters satisfying $\mathrm{p} \preceq \mathrm{q}$ constructively. It holds constructively that* $\text{tc-p} \preceq \text{tc-q}$ *and* $\text{mod-p} \preceq \text{mod-q}$. *Similarly for the relation $\preceq^*$.*

Proof. Let c be the constant such that $\mathrm{p}(G) \leq \mathrm{q}(G) + c$ for all graphs G. We compute that $\text{tc-p}(G) = \mathrm{p}(G/\Pi_{tc}(G)) \leq \mathrm{q}(G/\Pi_{tc}(G)) + c = \text{tc-q}(G) + c$ and $\text{mod-p}(G) = \max(1, \max_{H \in \mathcal{H}_p(G)} \mathrm{p}(H)) \leq \max(1, \max_{H \in \mathcal{H}_p(G)} \mathrm{q}(H)) + c = \text{mod-q}(G) + c$, hence $\text{tc-p} \preceq \text{tc-q}$ and $\text{mod-p} \preceq \text{mod-q}$. For $\preceq^*$, the inequalities follow very similarly by additionally using that the occurring quotient graphs have at most as many vertices as the original graph. □

Additionally, we show that the modular-variant is at least as expressive as the twinclass-variant and that the twinclass-variant is at least as expressive as the original parameter.

Lemma 2.4.22. *Let* p *be a graph parameter such that* $\mathrm{p}(H) \leq \mathrm{p}(G)$ *constructively whenever* H *is an induced subgraph of* G*. We have for every graph* G *that* $\text{tc-p}(G) \leq \mathrm{p}(G)$ *constructively. If* $\mathcal{H}_p(G) \neq \emptyset$*, we also have* $\text{mod-p}(G) \leq \text{tc-p}(G)$ *constructively. In particular,* $\text{mod-p} \preceq \text{tc-p} \preceq \mathrm{p}$.

Proof. Since $G/\Pi_{tc}(G)$ is an induced subgraph of G, we have that $\text{tc-p}(G) = \mathrm{p}(G/\Pi_{tc}(G)) \leq \mathrm{p}(G)$. For $\text{mod-p}(G) \leq \text{tc-p}(G)$, we argue that every graph in $\mathcal{H}_p(G)$ is an induced subgraph of $G/\Pi_{tc}(G)$. Consider any prime node $M \in \mathcal{M}_{\text{tree}}(G)$. Since M is a module, one can first see that all twinclasses X of $G[M]$ are of the form $X = C \cap M$, where C is some twinclass of G. This shows that $G[M]/\Pi_{tc}(G[M])$ is an induced subgraph of $G/\Pi_{tc}(G)$. Furthermore, since every twinclass is also a module and $\Pi_{mod}(G[M])$ consists of all inclusion-maximal proper modules of $G[M]$ by Theorem 2.4.7, we see that $\Pi_{tc}(G[M])$ must be a finer partition than $\Pi_{mod}(G[M])$. Hence, $G[M]/\Pi_{mod}(G[M])$ is an induced subgraph of $G[M]/\Pi_{tc}(G[M])$ and also of $G/\Pi_{tc}(G)$.

To establish $\text{mod-p} \preceq \text{tc-p}$, it remains to consider the case $\mathcal{H}_p(G) = \emptyset$. In that case, $\text{mod-p}(G) = 1$ is simply some constant independent of G which can be accounted for with an appropriate constant in the definition of $\text{mod-p} \preceq \text{tc-p}$. □

We need separate proofs for twinclass-modulators to obtain analogs of Lemma 2.4.14 and Corollary 2.4.18.

Lemma 2.4.23. *For any graph* G *and constant* $c \in \mathbb{N}_{>0}$*, we have constructively that* $\text{tc-tw}(G) \preceq \text{tcm}_{\text{tw} \leq c}(G) + c$ *and* $\text{tc-td}(G) \leq \text{tcm}_{\text{tw} \leq c}(G) + (c+1)\log_2 |V(G/\Pi_{tc}(G))|$*, hence* $\text{tc-tw} \preceq \text{tcm}_{\text{tw} \leq c}$ *and* $\text{tc-td} \preceq^* \text{tcm}_{\text{tw} \leq c}$.

Proof. Let $\mathcal{M}$ be a twinclass-modulator to treewidth c of G. We have that $G/\Pi_{tc}(G) - \mathcal{M}$ is an induced subgraph of $G - \bigcup(\mathcal{M})$, so $\text{tw}(G/\Pi_{tc}(G) - \mathcal{M}) \leq \text{tw}(G - \bigcup(\mathcal{M})) \leq c$, where we interpret $\mathcal{M}$ as a vertex subset of $G/\Pi_{tc}(G)$. Therefore, we can view $\mathcal{M}$ as a modulator to treewidth c of $G/\Pi_{tc}(G)$. By applying Lemma 2.4.14, we obtain that $\text{tc-tw}(G) = \text{tw}(G/\Pi_{tc}(G)) \leq |\mathcal{M}| + c$, proving the first inequality. By applying Corollary 2.4.18, we obtain $\text{tc-td}(G) = \text{td}(G/\Pi_{tc}(G)) \leq |\mathcal{M}| + (c+1)\log_2 |V(G/\Pi_{tc}(G))|$, thus proving the second inequality and $\text{tc-td} \preceq^* \text{tcm}_{\text{tw} \leq c}$ follows due to $|V(G/\Pi_{tc}(G))| \leq |V(G)|$. □

Lemma 2.4.24. *Let* $\mathcal{F}$ *be a graph family closed under induced subgraphs. For any graph* G*, it holds constructively that* $\text{tcm}_{\mathcal{F}}(G) \leq \text{dist}_{\mathcal{F}}(G)$*, hence* $\text{tcm}_{\mathcal{F}} \preceq \text{dist}_{\mathcal{F}}$.

Proof. Let X be an $\mathcal{F}$-modulator of G and consider the twinclass-modulator $\mathcal{M}$ obtained by taking every twinclass of G that X intersects. Since $G - \bigcup(\mathcal{M})$ is an induced subgraph of $G - X \in \mathcal{F}$, it follows by assumption that $\mathcal{M}$ is an $\mathcal{F}$-twinclass-modulator. Clearly, the size of $\mathcal{M}$ is at most the size of X. □

Lemma 2.4.25. *For every graph G and $c \in \mathbb{N}$, it holds that* $\mathrm{tw}(G) \leq \mathrm{dist}_{\mathrm{tw} \leq c}(G) + c$, $\text{tc-tw}(G) \leq \mathrm{tcm}_{\mathrm{tw} \leq c}(G) + c$, *and* $\mathrm{tcm}_{\mathrm{tw} \leq c}(G) \leq \mathrm{dist}_{\mathrm{tw} \leq c}(G)$, *hence* $\mathrm{tw} \preceq \mathrm{dist}_{\mathrm{tw} \leq c}$, *and* $\text{tc-tw} \preceq \mathrm{tcm}_{\mathrm{tw} \leq c}$, *and* $\mathrm{tcm}_{\mathrm{tw} \leq c} \preceq \mathrm{dist}_{\mathrm{tw} \leq c}$.

Proof. For the first inequality, we add the modulator into every bag of a tree decomposition of the remaining graph. The second inequality follows by applying the same argument on the level of the quotient graph. The third inequality follows from Lemma 2.4.24. □

Having lifted the known parameter relationships via Lemma 2.4.21 and Lemma 2.4.22 to the twinclass-variants and modular-variants, we will now see that appropriate versions of clique-width are at least as expressive as the twinclass- and modular-variants of pathwidth and treewidth.

Lemma 2.4.26 ([123])**.** *For any graph G, it holds constructively that* $\text{lin-cw}(G) \leq \text{tc-pw}(G) + 3$, *hence* $\text{lin-cw} \preceq \text{tc-pw}$.

Corollary 2.4.27. *For any graph G, we have* $\mathrm{cw}(G) \leq \text{mod-pw}(G) + 2$ *constructively, hence* $\mathrm{cw} \preceq \text{mod-pw}$.

Proof. Given a path decomposition of width k for every prime quotient graph $H \in \mathcal{H}_p(G)$, we construct a clique-expression μ for G using at most $k+2$ labels. To do so, we inductively construct a $(k+2)$-expression μ_M for every $G[M]$, $M \in \mathcal{M}_{\mathrm{tree}}(G)$.

As the base case, we consider the leaves of the modular decomposition tree which correspond to singleton modules $\{v\}$, $v \in V$, and therefore each $\mu_{\{v\}}$ simply consists of a single introduce-operation. For any internal node M of the modular decomposition tree with $\Pi_{mod}(G[M]) = \{M_1, \ldots, M_\ell\}$, we inductively assume that the clique-expressions $\mu_i := \mu_{M_i}$ for $G[M_i]$, $i \in [\ell]$, have already been constructed. Furthermore, we assume without loss of generality that every μ_i relabels all labels to label 1 at the end. We now distinguish between the node type of M in the modular decomposition tree. If M is a parallel node, then we obtain μ_M by successively taking the union of all μ_i, $i \in [\ell]$.

If M is a series node, then we set $\mu'_1 := \mu_1$ and $\mu'_{i+1} := \rho_{2\to1}(\eta_{1,2}(\mu'_i \oplus \rho_{1\to2}(\mu_{i+1})))$ for all $i \in [\ell - 1]$ and $\mu_M = \mu'_\ell$. So, we add one child module after the other and add all edges to the previous child modules using two labels.

If M is a prime node, then $G^q_M = G[M]/\Pi_{mod}(G[M]) \in \mathcal{H}_p(G)$ and we are given a path decomposition of G^q_M of width at most k. By Lemma 2.4.19, we can convert this path decomposition into a $(k+2)$-expression μ^q_M of G^q_M. We obtain the clique-expression μ_M by replacing every introduce-operation $\ell(v^q_{M_i})$ in μ^q_M with the $(k+2)$-expression $\rho_{1\to\ell}(\mu_i)$. This works, because by the definition of G^q_M and modules $G[M]$ can be constructed by substituting $G[M_i]$ for $v^q_{M_i}$ in G^q_M for all $M_i \in \Pi_{mod}(G[M])$. □

Corollary 2.4.28. *For every graph $G = (V, E)$, it holds constructively that* $\mathrm{mcw}(G) \leq \text{mod-tw}(G) + 3$ *and hence* $\mathrm{mcw} \preceq \text{mod-tw}$.

Proof. The proof works by the same principle as the proof of Corollary 2.4.27. Due to Lemma 2.4.20, a tree decomposition of width k can be transformed into a multi-clique-expression with $k+2$ labels. However, the resulting multi-clique-expression may contain deletion-operations ϵ_ℓ and we introduce another label $k+3$ and replace such deletions by

relabels $\rho_{\ell\to(k+3)}$. Thus, we obtain a multi-clique-expression using $k+3$ labels without any deletion-operations and by adding further relabel-operations we can again assume that all vertices have the same label at the end. This allows us to mimic the proof of Corollary 2.4.27 to obtain a multi-$(k+3)$-expression for G given tree decompositions of width k for every $H \in \mathcal{H}_p(G)$. □

Finally, we show that for clique-width and multi-clique-width, lifting to the twinclass- or modular-variants does not increase the expressivity of the parameters.

Lemma 2.4.29. *For every graph G, the following inequalities hold constructively:*

$$\mathrm{cw}(G) \le \max(2, \text{tc-cw}(G)), \qquad \mathrm{cw}(G) \le \max(2, \text{mod-cw}(G)),$$
$$\mathrm{mcw}(G) \le \max(2, \text{tc-mcw}(G)), \qquad \mathrm{mcw}(G) \le \max(2, \text{mod-mcw}(G)).$$

In particular, we have $\mathrm{cw} \preceq \text{mod-cw} \preceq \text{tc-cw} \preceq \mathrm{cw}$ *and* $\mathrm{mcw} \preceq \text{mod-mcw} \preceq \text{tc-mcw} \preceq \mathrm{mcw}$.

Proof. For the twinclass-variants, we see that any (multi-)clique-expression μ for $G/\Pi_{tc}(G)$ can be easily turned into a (multi-)clique-expression μ' for G using at most one extra label by replacing every introduce-operation in μ by a 2-expression creating the appropriate twinclass. The second label is only needed in the case of true twinclasses.

For the modular-variants, we use the same principle as in the proofs of Corollary 2.4.27 and Corollary 2.4.28. For parallel and series nodes, the quotient graph is an independent set or clique, respectively, and both cases can be handled by a simple 2-expression. For prime nodes, we are given a (multi-)k-expression of the quotient graph. In all three cases, we substitute inductively computed expressions for the introduce-operations contained in the expression for the quotient graph to obtain a (multi)-k-expression for the subgraph induced by the current module and can continue with the induction.

The final part follows from the proven inequalities together with Lemma 2.4.22. □

Counterexamples for Parameter Relationships

Due to the large number of considered parameters, we do not provide exhaustive comparisons between parameters p, q to show that $\mathrm{p} \preceq \mathrm{q}$ or $\mathrm{p} \preceq^* \mathrm{q}$ does *not* hold or even show that p and q are *incomparable*[7]. However, we emphasize that such results can often be transferred to twinclass- or modular-variants by using similar parameter-preserving transformations as in the proof of Lemma 2.4.11 that destroy any twinclass or modular structure. For example, attaching a private degree-1 vertex to every vertex of a clique yields a family of graphs with unbounded modular-treewidth, but bounded linear-clique-width.

We also provide an example family to show that the twinclass-treewidth can be arbitrarily larger than the modular-treewidth. Consider cliques where every vertex is replaced with a P_4, i.e., a path on 4 vertices, such that the path is a module in the new graph. Formally, for $k \in \mathbb{N}$, define the graph $G_k = (V_k, E_k)$ with $V_k = \{v_{i,j} : i \in [k], j \in [4]\}$ and $E_k = \{\{v_{i,j}, v_{i',j'}\} : i \neq i' \in [k], j, j' \in [4]\} \cup \{\{v_{i,j}, v_{i,j+1}\} : i \in [k], j \in [3]\}$. Since the path on 4 vertices is a prime graph, it is easy to see that G_k has no nontrivial twinclasses,

[7] Two graph parameters p, q are incomparable, if there exist graph families $\mathcal{F}_\mathrm{p}$ and $\mathcal{F}_\mathrm{q}$ such that p is bounded on $\mathcal{F}_\mathrm{p}$ and q is unbounded on $\mathcal{F}_\mathrm{p}$, and vice versa for $\mathcal{F}_\mathrm{q}$.

hence $\text{tc-tw}(G_k) = \text{tw}(G_k) \geq k-1$ because G_k contains a k-clique. However, we have $\text{mod-tw}(G_k) = \text{mod-pw}(G_k) = 1$, since the modular decomposition of G_k only contains series nodes and prime nodes whose associated quotient graphs are P_4's.

Search Game Characterizations

To prove that the graphs in our lower bound constructions have small treewidth or pathwidth, it is easier to use a *search game* characterization instead of directly constructing a tree or path decomposition.

Cops and Robbers. The search game corresponding to treewidth is the *omniscient-cops-and-robber-game*. In this game, $k+1$ cops try to capture a robber on a graph G. The cops can occupy vertices of G and the robber can run at infinite speed along edges through vertices that are not occupied by cops. The cops and the robber are omniscient in that they know each other's position at all times. At the start, the cops are placed on some vertices and the robber chooses a starting vertex for the escape. In every round, some cops get into their helicopters and declare where they will land next. Before these cops land again, the robber may move to any other vertex that is reachable by a path that does not contain any non-airborne cop. Afterwards, the cops land, and the next round begins. The cops *win* if they have a *strategy* to capture the robber by landing on the robber's location after a finite number of rounds; otherwise the robber wins.

Theorem 2.4.30 ([166])**.** *A graph G has treewidth at most k if and only if $k+1$ cops can win the omniscient-cops-and-robber-game on G.*

Mixed-search. The search game corresponding to pathwidth is the *mixed-search-game*. In such a game, the graph G represents a system of tunnels where all edges are contaminated by gas. The objective is to clear all edges of this gas. An edge can be cleared by either placing searchers at both of its endpoints or by moving a searcher along the edge. If there is a path from an uncleared edge to a cleared edge without any searchers on the vertices or edges of the path, then the cleared edge is recontaminated. A *search strategy* is a sequence of operations of the following types: a searcher can be placed on or removed from a vertex, and a searcher on a vertex can be moved along an incident edge and placed on the other endpoint. A search strategy is *winning* if after its termination all edges are cleared. The *mixed-search-number* of a graph G, denoted $\text{ms}(G)$, is the minimum number of searchers required for a winning strategy of the mixed-search-game on G.

Lemma 2.4.31 ([168])**.** *We have that* $\text{pw}(G) \leq \text{ms}(G) \leq \text{pw}(G)+1$.

3 Techniques

In this chapter, we discuss two techniques that are central to this thesis. First, we give an introduction to the cut-and-count technique, which is used to obtain parameterized algorithms for connectivity problems, in Section 3.1. Secondly, we give in Section 3.2.1 an overview of the construction principle of Lokshtanov et al. [126] for proving lower bounds relative to width parameters assuming SETH. This exposition is mostly independent of the considered width parameter and already gives an outlook on what considerations must be done to obtain such lower bounds. As a simple example, we give a tight lower bound for TOTAL DOMINATING SET[cutwidth] and show how this gives rise to a tight lower bound for DOMINATING SET[twinclass-cutwidth].

3.1 Connectivity Problems and Cut-and-Count

3.1.1 Connectivity Problems

The natural approach to solving connectivity problems by dynamic programming relative to a width parameter is to store the interaction of the *connectivity pattern* of partial solutions with parameter-dependent separators. We expand on this for the case of pathwidth and treewidth. Consider a connected undirected graph $G = (V, E)$, a separation (A, B) of G, and a vertex subset $X \subseteq V$. We want to store appropriate data about $G[X \cap A]$ and $G[X \cap B]$ at the vertex separator $A \cap B$ and based on this data check if $G[X \cap A]$ and $G[X \cap B]$ combine to a connected graph $G[X]$. Observe that even when $G[X]$ is connected, $G[X \cap A]$ and $G[X \cap B]$ may consist of several connected components. Let $C_1^A, \ldots, C_\ell^A$ denote the connected components of $G[X \cap A]$. The *connectivity pattern* of $G[X \cap A]$ at $A \cap B$ is the set family $\mathcal{C}^A = \{C_1^A \cap (A \cap B), \ldots, C_\ell^A \cap (A \cap B)\}$ which is a partition of $X \cap (A \cap B)$. In the same way, we obtain the connectivity pattern $\mathcal{C}^B$ of $G[X \cap B]$. One can see that $G[X]$ is connected if and only if the bipartite graph $(\mathcal{C}^A \cup \mathcal{C}^B, \{\{C_A, C_B\} : C_A \in \mathcal{C}^A, C_B \in \mathcal{C}^B, C_A \cap C_B \neq \emptyset\})$ is connected. If $|X \cap (A \cap B)| = k$, then the number of possible connectivity patterns is at most $k^k = 2^{k \log k}$ and at least $2^{\Omega(k \log k)}$, therefore this approach only leads to algorithms with running time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ given a tree decomposition of width k, which was long believed to be optimal.

3.1.2 Cut-and-Count-Technique

The cut-and-count-technique introduced by Cygan et al. [49, 51] reduces connectivity problems to locally checkable counting problems. With this technique, Cygan et al. have for the first time obtained single-exponential algorithms for connectivity problems parameterized

by treewidth, which in many cases even yield the optimal base in the running time assuming SETH [50]. We will be applying the cut-and-count-technique for several other graph parameters in this thesis to obtain single-exponential algorithms. We give a thorough introduction to the cut-and-count technique. In this section $G = (V, E)$ always refers to a connected undirected graph.

Consistent Cuts. The main engine behind the cut-and-count-technique are the so-called *consistent cuts*. A cut (V_L, V_R) of an undirected graph $G = (V, E)$ is *consistent* if $u \in V_L$ and $v \in V_R$ implies $\{u, v\} \notin E$, i.e., $E(V_L, V_R) = \emptyset$. For any subset $X \subseteq V$, we define $\texttt{cuts}_G(X) = \{(X_L, X_R) : (X_L, X_R)$ is a consistent cut of $G[X]\}$. Furthermore, a *consistently cut subgraph* of G is a pair $(X, (X_L, X_R))$ with $X \subseteq V$ and $(X_L, X_R) \in \texttt{cuts}_G(X)$. The set of all consistently cut subgraphs of G is denoted by $\mathcal{C}(G)$. Any non-trivial consistent cut (V_L, V_R), where non-trivial means $V_L \neq \emptyset \neq V_R$, of G is a locally checkable certificate that G is not connected. The following crucial lemma shows that the number of these certificates behaves nicely, even when G is not connected.

Lemma 3.1.1 ([51]). *For any $X \subseteq V$, it holds that $|\texttt{cuts}_G(X)| = 2^{\texttt{cc}(G[X])}$.*

Proof. Consider an arbitrary $(X_L, X_R) \in \texttt{cuts}_G(X)$ and any connected component C of $G[X]$. We must have $C \subseteq X_L$ or $C \subseteq X_R$, as otherwise an edge would cross the cut (X_L, X_R). Vice versa, any assignment of the connected components of $G[X]$ to the left and right side also yields a consistent cut of $G[X]$. Therefore, a consistent cut has two choices for every connected component of $G[X]$, which proves the lemma. □

Lemma 3.1.1 shows that we can decide whether $G[X]$ is connected or not by counting the number of consistent cuts of $G[X]$. In particular, we see that the number of consistent cuts of $G[X]$ is not divisible by 4 if and only if $G[X]$ is connected. Phrased more abstractly, the checking of the connectivity constraint can be replaced by counting locally checkable certificates (for non-connectivity), which is much simpler to implement in a dynamic programming routine relative to a width parameter. We proceed by explaining in more detail how we obtain a generic approach to solve connectivity problems in this way.

Overview. To solve a vertex selection problem on G involving a connectivity constraint, we make the following general definitions. The solutions to our problem are vertex sets and we denote the set of solutions by $\mathcal{S} \subseteq \mathcal{P}(V)$. The task is to determine whether $\mathcal{S}$ is empty or not. The *cut-and-count-technique* accomplishes this in two parts:

- **The Cut part:** By relaxing the connectivity constraint, we obtain a set $\mathcal{S} \subseteq \mathcal{R} \subseteq \mathcal{P}(V)$ of possibly connected solutions, called *candidates* which captures the non-connectivity related constraints of the problem. We pair up candidates $X \in \mathcal{R}$ with each of their consistent cuts (X_L, X_R) of $G[X]$ and obtain consistently cut subgraphs $(X, (X_L, X_R))$ which we collect in the set $\mathcal{Q}$.
- **The Count part:** We compute $|\mathcal{Q}|$ modulo 4 using a subprocedure. By Lemma 3.1.1 all disconnected candidates $X \in \mathcal{R} \setminus \mathcal{S}$ cancel, because they are consistent with a number of cuts that is divisible by 4. Hence, only connected candidates $X \in \mathcal{S}$ remain and we have $\mathcal{S} \neq \emptyset$ if the count $|\mathcal{Q}|$ is nonzero modulo 4.

The following theorem formalizes this approach.

Theorem 3.1.2. *Given families $\mathcal{R}, \mathcal{S} \subseteq \mathcal{P}(V)$, $\mathcal{Q} \subseteq \mathcal{P}(\mathcal{C}(G))$ satisfying $\mathcal{S} = \{X \in \mathcal{R} : G[X] \text{ is connected}\}$ and $\mathcal{Q} = \{(X, (X_L, X_R)) \in \mathcal{C}(G) : X \in \mathcal{R}\}$, it holds that $|\mathcal{Q}| \equiv_4 2|\mathcal{S}|$.*

Proof. Using Lemma 3.1.1, we compute

$$|\mathcal{Q}| = \sum_{X \in \mathcal{R}} 2^{\mathrm{cc}(G[X])} = \sum_{\substack{X \in \mathcal{R}: \\ \mathrm{cc}(G[X])=1}} 2^{\mathrm{cc}(G[X])} + \sum_{\substack{X \in \mathcal{R}: \\ \mathrm{cc}(G[X])\geq 2}} 2^{\mathrm{cc}(G[X])} = 2|\mathcal{S}| + 4 \cdot \sum_{\substack{X \in \mathcal{R}: \\ \mathrm{cc}(G[X])\geq 2}} 2^{\mathrm{cc}(G[X])-2}$$
$$\equiv_4 2|\mathcal{S}|. \qquad \square$$

Isolation. Theorem 3.1.2 has the following issue: it may happen that $\mathcal{S} \neq \emptyset$, but $|\mathcal{S}| \equiv_2 0$ and therefore $|\mathcal{Q}| \equiv_4 0$, so the approach cannot distinguish $\mathcal{S} \neq \emptyset$ from $\mathcal{S} = \emptyset$ in this case. We avoid this issue at the cost of randomization by using the isolation lemma.

Definition 3.1.3. A function $\mathbf{w}\colon U \to \mathbb{Z}$ *isolates* a set family $\mathcal{F} \subseteq \mathcal{P}(U)$ if there is a unique $S' \in \mathcal{F}$ with $\mathbf{w}(S') = \min_{S \in \mathcal{F}} \mathbf{w}(S)$.

Lemma 3.1.4 (Isolation Lemma, [134])**.** *Let $\mathcal{F} \subseteq \mathcal{P}(U)$ be a nonempty set family over a universe U. Let $N \in \mathbb{N}$ and for each $u \in U$ choose a weight $\mathbf{w}(u) \in [N]$ uniformly and independently at random. Then $\mathbb{P}[\mathbf{w} \text{ isolates } \mathcal{F}] \geq 1 - |U|/N$.*

Setting $U = V$ and given a weight function $\mathbf{w}\colon V \to [N]$, we partition the considered partial objects according to the *total weight* $\overline{w} \in [0, nN]$ of the contained objects, i.e., we define the families $\mathcal{R}^{\overline{w}} = \{X \in \mathcal{R} : \mathbf{w}(X) = \overline{w}\}$, $\mathcal{S}^{\overline{w}} = \{X \in \mathcal{S} : \mathbf{w}(X) = \overline{w}\}$, and $\mathcal{Q}^{\overline{w}} = \{(X, (X_L, X_R)) \in \mathcal{Q} : \mathbf{w}(X) = \overline{w}\}$ for every $\overline{w} \in [0, nN]$. If the sampled weight function $\mathbf{w}\colon V \to [N]$ isolates $\mathcal{S}$, then there exists a total weight $\overline{w} \in [0, nN]$ such that $|\mathcal{S}^{\overline{w}}| = 1$. Applying Theorem 3.1.2 to $\mathcal{R}^{\overline{w}}$, $\mathcal{S}^{\overline{w}}$, and $\mathcal{Q}^{\overline{w}}$ yields that $|\mathcal{Q}^{\overline{w}}| \equiv_4 2|\mathcal{S}^{\overline{w}}| \equiv_4 2 \not\equiv_4 0$ in this case. Thus, computing $|\mathcal{Q}^{\overline{w}}|$ modulo 4 allows us to determine whether $\mathcal{S}$ is empty or not when we have sampled an isolating weight function. As we do not know a priori which total weight $\overline{w}$ leads to $|\mathcal{S}^{\overline{w}}| = 1$, we simply have to try all of them.

Algorithm given Counting Procedure. Given a counting procedure $\mathbb{A}$, usually obtained via dynamic programming along a parameter-dependent decomposition, that computes $|\mathcal{Q}^{\overline{w}}|$ modulo 4 given a weight function $\mathbf{w}\colon V \to [N]$ and total weight $\overline{w} \in [0, nN]$, we decide whether $\mathcal{S}$ is nonempty as follows. We sample a weight function $\mathbf{w}\colon V \to [N]$ uniformly at random and compute $|\mathcal{Q}^{\overline{w}}|$ modulo 4 for *all* total weights $\overline{w} \in [0, nN]$. If $|\mathcal{Q}^{\overline{w}}| \not\equiv_4 0$ for *some* $\overline{w} \in [0, nN]$, then we return that $\mathcal{S}$ is not empty, otherwise we return that $\mathcal{S}$ is empty.

Error. We choose $N = 2|V| = 2n$ so that we obtain an error probability of at most $1/2$ in Lemma 3.1.4 (when $U = V$) and ensure that there is only a polynomial number of possible total weights. We argue that the resulting Monte Carlo algorithms only have one-sided errors in the sense that they cannot return false positives, but they can return false negatives with probability at most $1/2$. If $\mathcal{S} = \emptyset$, then $\mathcal{S}^{\overline{w}} = \emptyset$ and, by Theorem 3.1.2, $|\mathcal{Q}^{\overline{w}}| \equiv_4 0$ for all $\overline{w}$, so the algorithm correctly returns that $\mathcal{S}$ is empty. If $\mathcal{S} \neq \emptyset$, then the weight function $\mathbf{w}$ isolates $\mathcal{S}$ with probability at least $1/2$ by Lemma 3.1.4 and some $\overline{w}$ with $|\mathcal{S}^{\overline{w}}| = 1$ and hence $|\mathcal{Q}^{\overline{w}}| \not\equiv_4$ exists, so the algorithm correctly returns that $\mathcal{S}$ is nonempty.

Parity of Number of Solutions. We remark that Theorem 3.1.2 implies that the cut-and-count-technique also allows us to determine the parity of $|\mathcal{S}|$ by simply computing $|\mathcal{Q}|/2$ modulo two (without considering any weight function). So, the cut-and-count-technique essentially reduces the decision problem to solving several instances of the parity problem.

Complicated Universes. For some problems, e.g., CONNECTED ODD CYCLE TRANSVERSAL, the considered candidates and solutions are not simply vertex subsets, but carry additional data such as an appropriate coloring of the graph $G - X$ to facilitate the dynamic programming. In such cases, Theorem 3.1.2 does not directly apply, but the general approach remains the same. However, we need to adapt the universe U for the isolation lemma accordingly, so that the weights also take the additional data into account. As a side effect, $\mathcal{S}$ does not necessarily capture the kind of solutions described in the problem statement, but rather "augmented solutions". This also means that the cut-and-count-technique does not directly solve the standard parity variant of the problem as described in the previous paragraph.

Counting Connected Components. The standard cut-and-count-approach filters all candidates inducing at least two connected components by counting modulo $4 = 2^2$. Based on Lemma 3.1.1, we can also adapt the cut-and-count-technique to keep all candidates with at most ℓ connected components instead of only one by counting modulo $2^{\ell+1}$ instead of modulo 4, see Corollary 3.1.5. This variant will be used for FEEDBACK VERTEX SET to determine whether a graph is acyclic based on the number of vertices, edges, and connected components of the graph. In contrast, problems where the number of connected components must be maximized, such as CYCLE PACKING, do not allow for single-exponential algorithms relative to pathwidth under ETH as was shown by Cygan et al. [50].

Corollary 3.1.5. *Given $X \subseteq V$ and $\ell \in \mathbb{N}_{>0}$. It holds that $\mathtt{cc}(G[X]) \leq \ell$ if and only if $|\mathtt{cuts}_G(X)| \not\equiv_{2^{\ell+1}} 0$.*

Proof. By Lemma 3.1.1, there are exactly $2^{\mathtt{cc}(G[X])}$ consistent cuts of $G[X]$. Therefore, the number of consistent cuts is divisible by $2^{\ell+1}$ if $\mathtt{cc}(G[X]) > \ell$ and not divisible by $2^{\ell+1}$ if $\mathtt{cc}(G[X]) \leq \ell$. □

Fixing Vertices and Markers. Another common variant of the cut-and-count-technique is to fix some vertex $v_* \in V$ and only consider consistently cut subgraphs $(X, (X_L, X_R)) \in \mathcal{C}(G)$ with $v_* \in X_L$, which also implies that $v_* \in X$. To also consider solutions that do not contain v_*, one can first branch over all possible choices for v_*. The benefit of this variant is that for a fixed $v_* \in X \subseteq V$ there are exactly $2^{\mathtt{cc}(G[X])-1}$ such consistent cuts and hence one can work directly modulo two. The following lemma captures this approach.

Theorem 3.1.6 ([51]). *Given a fixed vertex $v_* \in V$ and families $\mathcal{R}, \mathcal{S} \subseteq \{X \subseteq V : v_* \in X\}$, $\mathcal{Q} \subseteq \mathcal{P}(\mathcal{C}(G))$ satisfying $\mathcal{S} = \{X \in \mathcal{R} : G[X] \text{ is connected}\}$ and $\mathcal{Q} = \{(X, (X_L, X_R)) \in \mathcal{C}(G) : X \in \mathcal{R}, v_* \in X_L\}$, it holds that $|\mathcal{Q}| \equiv_2 |\mathcal{S}|$.*

Proof. Due to fixing vertex v_* to the left side of any consistent cut, the whole connected component containing v_* must be contained in the left side. Following the proof of Lemma 3.1.1,

we see that there are $2^{\mathrm{cc}(G[X])-1}$ such consistent cuts of $G[X]$ for $X \in \mathcal{R}$. The statement follows via the analogous computation as in the proof of Theorem 3.1.2. □

Cygan et al. [51] pushed this technique further by considering candidate solutions with a dynamic number of *marked vertices* that must always be contained in the left side of the considered consistent cuts. In this way, the consistent cuts only count the connected components that do not contain any marked vertices and this can be used to also bound the number of connected components from above while still working modulo two. We prefer to avoid these variants, as fixing a vertex creates some extra edge cases in the formal definitions and dynamic programming recurrences. On top of that, the marker technique additionally complicates the universe.

Obtaining a Solution Set. This presentation of the cut-and-count-technique only determines whether a solution exists or not for the sake of simplicity instead of actually computing some solution $X \in \mathcal{S}$. Usually, it is possible to adapt the technique to compute a solution with polynomial overhead by repeatedly solving an annotated variant of the problem, where vertices can be forced into the considered candidates or forbidden, and updating the annotations between the iterations until X is completely given as an annotation. To keep the error probability in check, the weight function is only sampled once at the beginning, before any annotations are set, and used for all iterations.

Vertex Costs. To handle cardinality constraints or cost[1] constraints, when given vertex costs $\mathbf{c}\colon V \to \mathbb{N}_{>0}$, the dynamic programming algorithms to compute $|\mathcal{Q}|$ also partition the candidates according to their attained cost, so all table entries are in particular indexed by some cardinality/cost and some total weight. It is not sufficient to only consider candidates of, say, minimum cost, because these might cancel later on during the execution of the dynamic program and not lead to a connected solution. Consequently, the cut-and-count-technique cannot handle superpolynomial vertex costs without increasing the running time by a superpolynomial factor. Therefore, we will always only work with polynomially bounded costs when using the cut-and-count-technique.

3.1.3 Related Work and Techniques

We discuss some shortcomings of the cut-and-count-technique and what approaches and other techniques have been developed to deal with these shortcomings. As the cut-and-count-technique is a mainstay throughout this thesis, we discuss further related work when it is more appropriate in Part II and Part III.

Shortcomings of Cut-and-Count. The cut-and-count-technique has several shortcomings: first, due to its reliance on the isolation lemma, it can only yield randomized algorithms. Secondly, we must store separate counts for each possible size/cost of partial solutions in the dynamic programming, therefore the cut-and-count-technique can only deal with

[1] Throughout this thesis, we distinguish between *costs* and *weights*; costs always refer to the objective function and weights refer to the weight function used for the isolation lemma.

weighted problem variants at the cost of introducing a pseudo-polynomial dependence on the weights in the running time. Thirdly, when using the isolation lemma, the counts must also be separated according to the weights and since both size and weights can usually be at least linear in the number of vertices n, the polynomial factor of the running time is at least n^2. Finally, as we cannot precisely distinguish which contributions come from unconnected solutions and which from connected solutions, the cut-and-count-technique cannot solve counting variants (with the exception of counting modulo 2 in many cases).

Improving Upon Quadratic Running Time. As explained, use of the isolation lemma leads to at least a quadratic dependence on the number of vertices n. By relying on polynomial identity testing instead, Ziobro and Pilipczuk [177] show how to adapt the cut-and-count-technique for the case of HAMILTONIAN CYCLE[treewidth] so that the running time only depends subquadratically on n. More precisely, they obtain a Monte Carlo algorithm for HAMILTONIAN CYCLE[treewidth] with running time $4^k n(k \log n)^{\mathcal{O}(1)}$ by computing in a field of characteristic 2 and using the Schwartz-Zippel lemma.

Rank-Based Approach. At the cost of increasing the base of the exponent in the running time, the first three shortcomings can be dealt with by using the *rank-based approach* of Bodlaender et al. [19]. Naive dynamic programming can usually be understood as separating partial solutions into equivalence classes based on which other partial solutions they are *compatible* with and building partial solutions based on this information in a bottom-up fashion. For connectivity problems, this leads to the mentioned concept of *connectivity patterns*, which leads to $2^{\Omega(k \log k)}$ equivalence classes. The crucial observation of the rank-based approach is that it is not necessary to store all such equivalence classes. Instead, it suffices to keep a *representative set* $\mathcal{F}$ of partial solutions such that we have for every partial solution Y that if there exists some partial solution X (among all partial solutions) that is compatible with Y, then there also exists a partial solution $X' \in \mathcal{F}$ that is compatible with Y. Such representative sets can be found by computing a basis of an appropriate *compatibility matrix*, which is where the *rank* comes into play. However, the dimensions of the considered compatibility matrix are exponential in the parameter, and the basis computation is done via fast matrix multiplication, which is the bottleneck of the running time of the obtained algorithms and makes the running time base dependent on the value of the *matrix multiplication exponent*[2] ω. Fomin et al. [76, 77] reformulate the computation of a representative set in the rank-based approach for connectivity problems in matroid terms and give faster algorithms for its computation in special cases, leading to an improvement in the base of the running time, however the dependency on ω remains and the representative set computation remains the bottleneck.

Let us make some additional remarks on the rank-based approach here. Variants of the rank-based approach can also be used for problems without connectivity constraints, for example, Jansen and Nederlof [111] and Groenland et al. [95] obtain algorithms for coloring problems parameterized by cutwidth using similar methods. Moreover, if the slow computation of a basis can be avoided and performed by other means, via e.g. problem-specific tricks, then also tight upper bounds under SETH can be obtained with the rank-based

[2]The best currently known bound for ω is $\omega < 2.37286$ by Alman and Vassilevska W. [3].

approach, examples are given by the two works on colorings [111, 95] and an algorithm for HAMILTONIAN CYCLE[pathwidth] by Cygan et al. [48] and its counting variant by Curticapean et al. [45]. For more information on rank-based methods, we direct the reader to the survey of Nederlof [136].

Squared Determinant Approach. To deal with the final shortcoming of the cut-and-count-technique, Bodlaender et al. [19] have also shown how counting variants of connectivity problems parameterized by pathwidth or clique-width can be solved in single-exponential time. These algorithms are obtained via the *squared determinant approach*, which uses a reduction underlying Kirchhoff's Matrix Tree Theorem that allows counting of connected objects by computing appropriate determinants. The determinants are then squared so that their sign can be ignored in the computation. By using the Leibniz formula for the determinant, partial sums are obtained that can be evaluated by dynamic programming along a path/tree decomposition. The squared determinant approach does not suffer from a running time bottleneck like the rank-based approach, however the complexity of counting variants of connectivity problems can simply be higher than the complexity of the optimization or decision variant. For example, Curticapean et al. [45] have shown a $\mathcal{O}^*((6-\varepsilon)^{\mathrm{pw}})$ lower bound for counting Hamiltonian Cycles parameterized by pathwidth, which is tight as a $\mathcal{O}^*(6^{\mathrm{pw}})$-time algorithm is given by Bodlaender et al. [19] via the squared determinant approach. In comparison, the decision version has optimal base $2+\sqrt{2}$ by results of Cygan, Kratsch, and Nederlof [48].

Random Bits. The sampling of the weight function for the isolation lemma used in the cut-and-count-technique uses $\mathcal{O}(n \log n)$ random bits. Nederlof et al. [138] study *isolation schemes* that use fewer random bits for problems on *decomposable graphs*, e.g., graphs of low treewidth or low treedepth. For example, they prove that Hamiltonian Cycles in graphs of treewidth at most t can be isolated with $\mathcal{O}(t \log n + \log^2(n))$ random bits and that maximum-size independent sets in graphs of treedepth at most d can be isolated using $\mathcal{O}(d)$ random bits and polynomially bounded weights.

3.2 SETH-Lower Bounds Relative to Width Parameters

3.2.1 General Lower Bound Principle

In this section, we give an overview of the construction principle by Lokshtanov et al. [126] for fine-grained lower bounds relative to width parameters. Lokshtanov et al. [126] first obtained tight lower bounds for several benchmark problems parameterized by pathwidth and treewidth, e.g., unless SETH fails, there is no $\varepsilon > 0$ such that VERTEX COVER[treewidth] can be solved in time $\mathcal{O}^*((2-\varepsilon)^{\mathrm{tw}(G)})$ or such that DOMINATING SET[treewidth] can be solved in time $\mathcal{O}^*((3-\varepsilon)^{\mathrm{tw}(G)})$. A large number of lower bounds build upon this principle by now, including lower bounds for other width parameters such as cutwidth or clique-width. As we will apply this principle to several width parameters in this thesis, we give an overview independent of the considered width parameter.

Reduction Objectives. Let w denote the considered width parameter and suppose that we want to rule out running time $\mathcal{O}^*((\alpha-\varepsilon)^{\mathrm{w}(G)})$ for all $\varepsilon > 0$ for some *target problem* parameterized by w, assuming SETH. To achieve such lower bounds, we want to reduce q-SATISFIABILITY, where q is the maximum clause size, with n variables to an instance of the target problem of width roughly $n \log_\alpha(2)$. Essentially, we must simulate $1/\log_\alpha(2) = \log_2(\alpha)$ variables per unit of width. Having achieved such a reduction, an improved algorithm for the target problem with running time $\mathcal{O}^*((\alpha-\varepsilon)^{\mathrm{w}(G)})$ for some $\varepsilon > 0$ would imply that there is some $\varepsilon' > 0$ such that q-SATISFIABILITY can be solved in time $\mathcal{O}^*((2-\varepsilon')^n)$ for all q, thus violating SETH.

Grid-like Construction. The graphs obtained by the construction principle of Lokshtanov et al. [126] can be interpreted as a *matrix/grid of blocks*, where each block spans several rows and one column. Every row is a long *path-like gadget* that simulates a constant number ($\log_2(\alpha)$) of variables of the SATISFIABILITY instance and which contributes one unit of width. For technical reasons, we consider bundles of rows simulating a *variable group* of appropriate size when α is not a power of two. Every column corresponds to a clause and consists of gadgets that *decode* the states on the path gadgets and check whether the resulting assignment satisfies the clause via a *clause gadget*. Due to the grid-like construction, the resulting graph has a *linear* structure instead of a tree-like structure, meaning that the lower bounds already apply to a linear version of the width parameter, e.g., the lower bounds already hold for pathwidth instead of treewidth or they hold for linear-clique-width instead of clique-width. A schematic depiction of the grid-like construction is given in Figure 3.1.

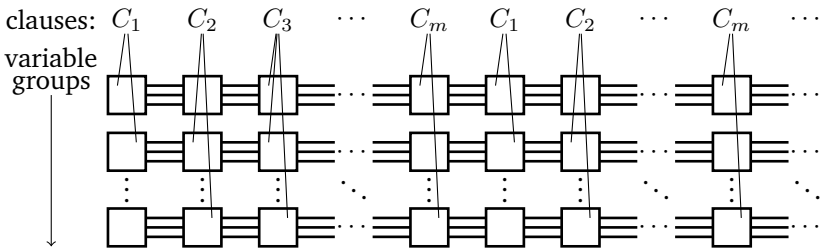


Fig. 3.1.: A schematic depiction of the grid-like lower bound construction. Each bundle of rows corresponds to a variable group, and each column to a clause. The connections between the columns depend on the considered width parameter. Each clause gadget accesses a subset of rows depending on which variable groups intersect the corresponding clause. We highlight that multiple columns correspond to the same clause.

Path Gadgets. Usually, the most challenging part of this construction principle is to construct appropriate *path gadgets* that lie at the intersection of each row and column. The design of the *decoding gadgets* is simpler or can often be adapted from known constructions. There is one path gadget at the intersection of each row and column. The goal is to construct a path gadget admitting as many states as possible for the target problem, since the number of such states corresponds to the base of the running time for which we obtain a lower bound, i.e., α states rule out all running time bases below α. Since every row should contribute one unit of width, adjacent path gadgets in a row must be connected by a separator of

size one where the type of separator depends on the considered width parameter, thus the path gadget design greatly depends on the considered parameter. We continue by giving a detailed explanation of the considerations that need to be made when designing a path gadget.

State Transitions. When proving that a solution to the target problem yields a satisfying assignment for SATISFIABILITY, we require that the path gadget state remains *stable* along each *row*. Otherwise, if the states were allowed to change, one could pick a satisfying assignment for each clause *separately*, which does not necessarily lead to an assignment satisfying *all* clauses simultaneously. Unfortunately, it is often not possible to construct path gadgets where the state remains stable. However, we can construct path gadgets where the states can only transition in a controlled way.

Controlling the State Transitions. Suppose that there is some *transition order* on the states, i.e., state i can transition to state j only if $i \leq j$; this in particular implies that the state can change only a finite number of times along each row as there is only a finite number of possible states of the path gadget. By making the rows long enough and repeating clauses, the pigeonhole principle allows us to find a region of columns spanning all clauses where the states on all rows remain stable, hence obtaining the same assignment for every clause. Therefore, our goal is to find a large set of states and an appropriate path gadget admitting such a transition order.

Determining a Transition Order. As the connection between adjacent path gadgets depends on the considered width parameter, so does the process of determining an appropriate transition order. Indeed, some state transitions are *forced* upon us due to the parameter-dependent separator connecting adjacent path gadgets. Here, we must start distinguishing between a state at a separator and a state of a whole path gadget; a *path gadget state* is usually an aggregate of *separator states* at the left and the right separator and an internal state. For sparse width parameters such as pathwidth, determining a transition order is simpler, because the number of possible separator states is very limited, and thus a less systematic approach often suffices. However, for dense width parameters, the possible set of separator states is much larger, and usually we need to select a specific subset of separator states to obtain path gadget states admitting a transition ordering. Therefore, we begin by analyzing the forced state transitions across a parameter-dependent separator. We obtain a *transition matrix* capturing for each separator state which other separator states it can transition into. The transition matrix can also be understood as a *compatibility matrix* showing which pairs of separator states, one on the left side of the separator and one on the right, can lead to a globally feasible solution and which cannot. After possibly reordering the rows and columns of the compatibility matrix, a transition order must induce a *triangular submatrix* with ones on the diagonal, where the ones on the diagonal later allow for the path gadget state to remain stable.

From Triangular Submatrix to Path Gadget Behavior. Having determined a large triangular submatrix, representing a transition ordering, of the compatibility matrix, we can

deduce the desired path gadget behavior as follows. We pair the separator states along the main diagonal, which consists of only ones, of the triangular submatrix; these obtained state pairs will model the path gadget states. A state pair is *attainable* by a path gadget, if there exists a partial solution such that the first state of the pair is communicated to one *boundary* of the path gadget and the second state to the other boundary. If there is a path gadget such that exactly the state pairs induced by the triangular submatrix are attainable, then the path gadget has the desired transition behavior: the ones on the diagonal show that the path gadget state can remain stable and the triangularity of the submatrix shows that state transitions can only occur in one direction, e.g., we can transition from path gadget state i to path gadget state j only if $i \leq j$.

Decoding and Clause Gadgets. We make some preliminary remarks on the decoding and clause gadgets. Each column of the construction contains several decoding gadgets, one for each row bundle corresponding to a variable group, which looks at the path gadgets in its associated row bundle and decodes their states into a truth assignment for the variable group. The clause gadget attaches to (some of) the decoding gadgets in its column and checks whether the decoded truth assignments satisfy the clause associated with the column. If the interface between the path and decoding gadgets is designed in a consistent way, then the decoding and clause gadgets can often be designed independently of the considered width parameter; we will see this later when proving various lower bounds for connectivity problems. The clause gadget is where the restriction q on the clause size comes into play, as it allows us to bound the size of the clause gadget by some function of q. However, there are many cases, in particular in this thesis, where the clause gadget size does not scale with the clause size, allowing us to prove the lower bounds even under the weaker assumption of CNF-SETH instead of SETH.

3.2.2 Lower Bound for Total Dominating Set[cutwidth]

As a simple example of the lower bound technique, we give a tight lower bound for Total Dominating Set[cutwidth] which we then lift to a tight lower bound for Dominating Set[tc-cutwidth]; these two lower bounds are based on joint work with Stefan Kratsch [102]. Total Dominating Set[treewidth] can be solved in time $\mathcal{O}^*(4^{\mathrm{tw}(G)})$ by van Rooij et al. [164]. We will prove that there is no $\varepsilon > 0$ such that Total Dominating Set[cutwidth] can be solved in time $\mathcal{O}^*((4-\varepsilon)^{\mathrm{ctw}(G)})$ unless SETH fails; due to the mentioned algorithm and Section 2.4.4 this result is tight. We show this by adapting the lower bound construction for Dominating Set given by Lokshtanov et al. [126] and the observation of van Geffen et al. [94] that slightly changing the construction already yields the same lower bound parameterized by cutwidth. Since 4 is a power of two, we do not need to group several rows to avoid rounding errors, hence several technicalities in the proof can be avoided. We begin with a preliminary analysis of states in the Total Dominating Set problem, which will serve as our guide in the formal construction and proof of the lower bound.

Vertex States. We begin by studying the possible states of a vertex with respect to a partial solution of Total Dominating Set; we also refer to vertex states as *atomic states* or *atoms*. In

$\mathbf{a}_x$ vs. $\mathbf{b}_y$	$\mathbf{1}_1$	$\mathbf{1}_0$	$\mathbf{0}_1$	$\mathbf{0}_0$
$\mathbf{1}_1$	1	1	1	1
$\mathbf{1}_0$	1	1	0	0
$\mathbf{0}_1$	1	0	1	0
$\mathbf{0}_0$	1	0	0	0

$\mathbf{a}_x$ vs. $\mathbf{b}_y$	$\mathbf{0}_0$	$\mathbf{1}_0$	$\mathbf{0}_1$	$\mathbf{1}_1$
$\mathbf{1}_1$	1	1	1	1
$\mathbf{1}_0$	0	1	0	1
$\mathbf{0}_1$	0	0	1	1
$\mathbf{0}_0$	0	0	0	1

Tab. 3.1.: The compatibility matrix for TOTAL DOMINATING SET[cutwidth], describing which vertex states are compatible across an edge. On the right, the columns of the compatibility matrix are rearranged so that it induces a triangular matrix.

this case, a partial solution is simply some vertex subset $X \subseteq V$. For some vertex $v \in V$, we should clearly distinguish whether $v \in X$ or $v \notin X$ and for sake of the domination property we should also distinguish whether $v \in N(X)$ or $v \notin N(X)$. Combining these two choices yields in total 4 possibilities, which we encode as $\mathbf{Atoms} = \{\mathbf{1}_1, \mathbf{1}_0, \mathbf{0}_1, \mathbf{0}_0\}$, where the bold number represents whether the vertex is contained in X and the subscript represents whether the vertex is dominated by X. We denote the state of v with respect to X by $\mathbf{state}_X(v) \in \mathbf{Atoms}$. So, X is a total dominating set of G if and only if $\mathbf{state}_X(v) \in \{\mathbf{0}_1, \mathbf{1}_1\}$ for all $v \in V$.

State Behavior Across Separators. Next, we consider the state behavior across the parameter-defining separators, which are simply edge separators/cuts for cutwidth. It is often sufficient to consider separators of size 1, so we simply consider a single edge. We define the relation $\sim$ on $\mathbf{States}$ as follows: $\mathbf{a}_x \sim \mathbf{b}_y$ if and only if $(x = 1 \vee \mathbf{b} = \mathbf{1}) \wedge (y = 1 \vee \mathbf{a} = \mathbf{1})$, leading to the first *compatibility matrix*[3] in Table 3.1. The relation $\sim$ captures the following situation. Given two graphs $G_i = (V_i, E_i)$, $i \in [2]$, with a distinguished vertex $\hat{v}_i \in V_i$ and partial solution $X_i \subseteq V_i$ such that $\mathbf{state}_{X_i}(V_i \setminus \{\hat{v}_i\}) \subseteq \{\mathbf{0}_1, \mathbf{1}_1\}$, let $\widetilde{G}$ denote the graph obtained by adding the edge $\{\hat{v}_1, \hat{v}_2\}$ in the (disjoint) union of G_1 and G_2. We want that $\mathbf{state}_{X_1}(\hat{v}_1) \sim \mathbf{state}_{X_2}(\hat{v}_2)$ if and only if $X_1 \cup X_2$ is a total dominating set of $\widetilde{G}$. Since all remaining vertices of $\widetilde{G}$ are already dominated, it suffices to check that $\hat{v}_i$ is already dominated by X_i or newly dominated via the edge $\{\hat{v}_1, \hat{v}_2\}$, which is precisely implemented by the relation $\sim$.

Transition Ordering. The compatibility matrix guides us in deciding which vertex states should be implemented at the boundary of the path gadget to obtain path gadget states that can only transition according to some appropriate transition ordering. Reordering the columns in the compatibility matrix, we obtain the second matrix in Table 3.1 which is triangular. By pairing the vertex states along the main diagonal, we obtain the ordered pairs (first column state, then row state)

$$\mathbf{s}^1 = (\mathbf{0}_0, \mathbf{1}_1), \mathbf{s}^2 = (\mathbf{1}_0, \mathbf{1}_0), \mathbf{s}^3 = (\mathbf{0}_1, \mathbf{0}_1), \mathbf{s}^4 = (\mathbf{1}_1, \mathbf{0}_0) \in \mathbf{Atoms}^2.$$

We say that a path gadget P with *entry vertex* p_{entry} and *exit vertex* p_{exit} can *attain state* $(\mathbf{b}_y, \mathbf{a}_x) \in \mathbf{Atoms}^2$ if there is some $X \subseteq V(P)$ such that $(\mathbf{state}_X(p_{entry}), \mathbf{state}_X(p_{exit})) = (\mathbf{b}_y, \mathbf{a}_x)$ and $\mathbf{state}_X(V(P) \setminus \{p_{entry}, p_{exit}\}) \subseteq \{\mathbf{0}_1, \mathbf{1}_1\}$. If we can build a path gadget that can attain the states $\mathbf{s}^1, \mathbf{s}^2, \mathbf{s}^3, \mathbf{s}^4$ and no other states, then the triangular compatibility matrix

[3] The compatibility matrices for larger separators often turn out to be *Kronecker powers* of the compatibility matrix for separators of size 1, e.g., see Groenland et al. [95]. In such cases, the essential matrix structure is already captured by the size-1 case, which hence serves as a good starting point.

tells us that between two consecutive path gadgets state $\mathbf{s}^i$ can only transition to $\mathbf{s}^j$ with $i \leq j$, as the path gadgets are connected by adding an edge between the exit vertex of the first gadget and the entry vertex of the second. Hence, we obtain the transition ordering $\mathbf{s}^1 \preceq \mathbf{s}^2 \preceq \mathbf{s}^3 \preceq \mathbf{s}^4$, which will allow us to obtain a satisfying assignment from a solution to the TOTAL DOMINATING SET instance as discussed before.

From State Analysis to Construction. After the state analysis, we now have a concrete guideline for the behavior that should be implemented by the path gadget. An additional design constraint to consider for problems with a cardinality/cost constraint, in particular TOTAL DOMINATING SET, is that all states attainable by the path gadget should have equal cost. Otherwise, choosing a state with lower cost frees up budget that could be used for other gadgets to achieve undesired behavior which could cause the reduction proof to fail. Besides the path gadgets, we also need to design decoding and clause gadgets. However, for the lower bounds in this thesis relative to width parameters, the design of the decoding and clause gadgets is considerably simpler and can often be directly adapted from already-known constructions. We now proceed with the formal construction and proof of the TOTAL DOMINATING SET[cutwidth] lower bound.

Theorem 3.2.1. *There is no algorithm solving* TOTAL DOMINATING SET *in time* $\mathcal{O}^*((4-\varepsilon)^{\mathrm{ctw}(G)})$ *for some* $\varepsilon > 0$*, unless SETH is false.*

Construction Setup. Let σ be a q-SATISFIABILITY instance with n variables and m clauses. We construct a graph $G = G(\sigma)$ of cutwidth $n/2 + 4q + \mathcal{O}(1)$. We assume without loss of generality that σ has an even number of variables and partition the variables into pairs, so that the i-th pair, $i \in [n/2]$, consists of the variables x_{2i-1} and x_{2i}.

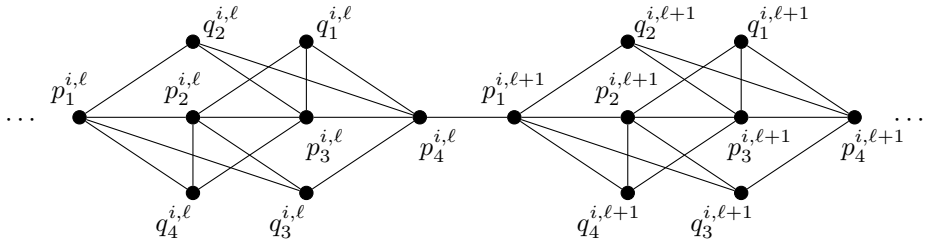


Fig. 3.2.: Two path segments $P^{i,\ell}$, $P^{i,\ell+1}$ and guards $Q^{i,\ell}$, $Q^{i,\ell+1}$.

Path Gadgets. The design of the path gadget for TOTAL DOMINATING SET[cutwidth] is relatively simple, one can observe that a reasonable total dominating set on a long path should repeatedly take two consecutive vertices with a gap of two. Such a total dominating set essentially cycles between the considered four vertex states, or atoms, on the path and we design our path gadget based on this observation. For every pair $i \in [n/2]$, we create a path P^i consisting of $4m(3\frac{n}{2}+1)$ vertices and partition the path into $m(3\frac{n}{2}+1)$ segments consisting of four vertices. Each segment corresponds to one path gadget and we have $3\frac{n}{2}+1$ column regions which cover the m clauses each. Segment $\ell \in [m(3\frac{n}{2}+1)]$ consists of the vertices $P^{i,\ell} = \{p_1^{i,\ell}, p_2^{i,\ell}, p_3^{i,\ell}, p_4^{i,\ell}\}$; $p_1^{i,\ell}$ is the *entry vertex* and $p_4^{i,\ell}$ is the *exit vertex*. To each

segment ℓ, we attach four *guard vertices* $Q^{i,\ell} = \{q_1^{i,\ell}, \ldots, q_4^{i,\ell}\}$ such that $N(q_j^{i,\ell}) = P^{i,\ell} \setminus \{p_j^{i,\ell}\}$ for all $j \in [4]$, i.e., for every vertex on $P^{i,\ell}$ there is one guard that avoids this vertex and is only adjacent to all other vertices on $P^{i,\ell}$, see Figure 3.2. The guards $Q^{i,\ell}$ ensure that a total dominating set must contain at least two vertices of $P^{i,\ell}$.

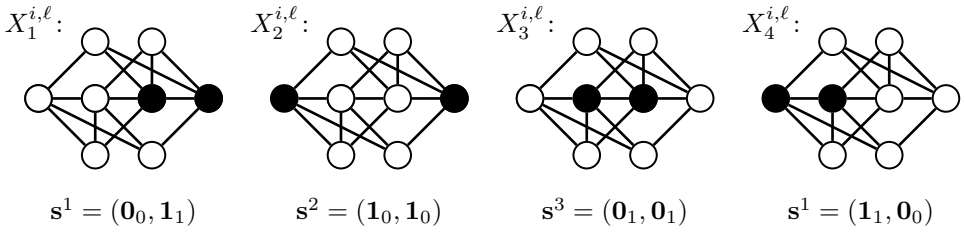


Fig. 3.3.: The four path gadget states $X_1^{i,\ell}$, $X_2^{i,\ell}$, $X_3^{i,\ell}$, and $X_4^{i,\ell}$ on the path segment $P^{i,\ell}$. The filled vertices belong to the corresponding set.

States of the Path Gadget. Let $X_1^{i,\ell} = \{p_3^{i,\ell}, p_4^{i,\ell}\}$, $X_2^{i,\ell} = \{p_1^{i,\ell}, p_4^{i,\ell}\}$, $X_3^{i,\ell} = \{p_2^{i,\ell}, p_3^{i,\ell}\}$, $X_4^{i,\ell} = \{p_1^{i,\ell}, p_2^{i,\ell}\}$, see Figure 3.3 for a depiction of these sets. Note that these sets attain the states $\mathbf{s}^1$, …, $\mathbf{s}^4$ as desired, e.g., $X_1^{i,\ell}$ attains the path gadget state $\mathbf{s}^1 = (\mathbf{0}_0, \mathbf{1}_1)$ as the entry vertex $p_1^{i,\ell}$ has vertex state $\mathbf{0}_0$ and the exit vertex $p_4^{i,\ell}$ the vertex state $\mathbf{1}_1$ with respect to $\mathbf{1}_i^\ell$.

Decoding Gadget. For each segment $P^{i,\ell}$, we introduce four *connector vertices* $\hat{z}_j^{i,\ell}$, $j \in [4]$ and four *clique vertices* $z_j^{i,\ell}$, $j \in [4]$, together with two *clique guards* $y_1^{i,\ell}$ and $y_2^{i,\ell}$. The connector vertex $\hat{z}_j^{i,\ell}$ is adjacent to $P^{i,\ell} \setminus X_j^{i,\ell}$ and the clique vertex $z_j^{i,\ell}$. The clique guards $y_1^{i,\ell}$ and $y_2^{i,\ell}$ are adjacent and the clique vertices $z_j^{i,\ell}$, $j \in [4]$, together with $y_1^{i,\ell}$ form a clique. The function of the decoding gadget is that the clique vertices decode which path gadget state was chosen on $P^{i,\ell}$ and serve as a simple interface for the clause gadget.

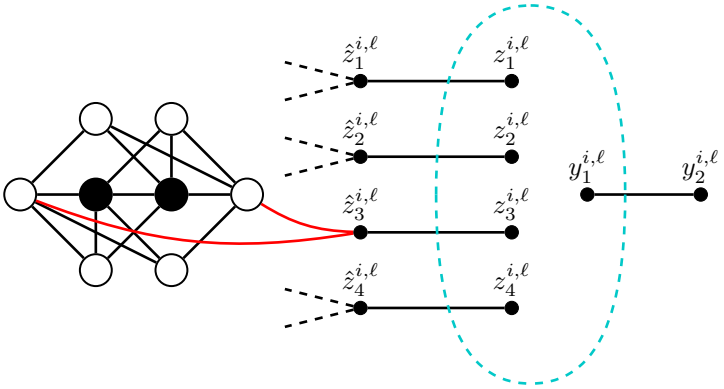


Fig. 3.4.: The block gadget B_i^ℓ. The vertices in the dashed ellipse form a clique. The vertex $\hat{z}_3^{i,\ell}$ is adjacent to $P^{i,\ell} \setminus X_3^{i,\ell}$.

Blocks and Regions. We define $Z^{i,\ell} = \{z_j^{i,\ell}, \hat{z}_j^{i,\ell} : j \in [4]\} \cup \{y_1^{i,\ell}, y_2^{i,\ell}\}$, $B^{i,\ell} = P^{i,\ell} \cup Q^{i,\ell} \cup Z^{i,\ell}$, and $R^\gamma = \bigcup_{i=1}^{n/2} \bigcup_{\ell=(\gamma-1)m+1}^{\gamma m} B^{i,\ell}$ for $\gamma \in [3\frac{n}{2} + 1]$. The $B^{i,\ell}$ are called *blocks*; the R^γ are called *regions*. See Figure 3.4 for a depiction of a single block. Each region can be

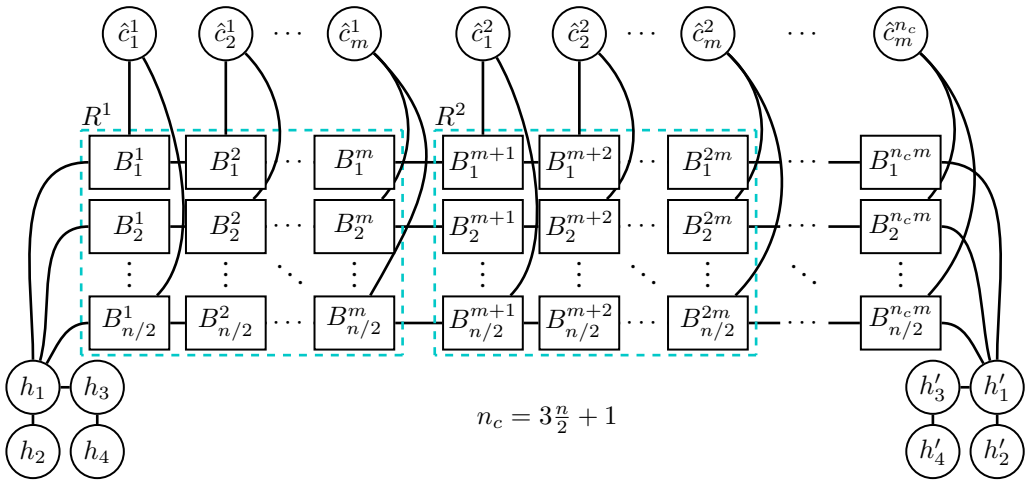


Fig. 3.5.: G viewed as a matrix of block gadgets. The dashed rectangles delineate different regions.

visualized as a matrix of blocks with $n/2$ rows and m columns and we have $3\frac{n}{2}+1$ regions in total, see Figure 3.5. Notice that there is a matching of size $n/2$ between the exit vertices of column ℓ and the entry vertices of column $\ell+1$.

Clause Gadgets. We fix a bijection $\kappa\colon \{0,1\}^2 \to [4]$ mapping truth assignments of a pair of variables to a path gadget state, which is indicated by a number between 1 and 4. For each clause $C_{\ell'}$, $\ell' \in [m]$, we introduce $3\frac{n}{2}+1$ *clause vertices* $\hat{c}_{\ell'}^{\gamma}$, $\gamma \in [3\frac{n}{2}+1]$, one per region. The vertex $\hat{c}_{\ell'}^{\gamma}$ is adjacent to some vertices in the blocks $B^{i,(\gamma-1)m+\ell'}$ for every $i \in [n/2]$. Namely, for every variable pair i, we add an edge between $\hat{c}_{\ell'}^{\gamma}$ and $z_j^{i,(\gamma-1)m+\ell'}$ if $\kappa^{-1}(j)$ is a truth assignment for variable pair i that satisfies clause $C_{\ell'}$. Hence, in a fixed column $(\gamma-1)m+\ell'$, all clique vertices that correspond, via κ, to truth assignments of some variable pair $i \in [n/2]$ satisfying clause $C_{\ell'}$ are adjacent to the clause vertex $\hat{c}_{\ell'}^{\gamma}$.

Endpoint Guards. Finally, we introduce eight *endpoint guards* h_t, h_t', $t \in [4]$, together with the edges $\{h_1,h_2\}$, $\{h_1,h_3\}$, $\{h_3,h_4\}$, $\{h_1',h_2'\}$, $\{h_1',h_3'\}$, $\{h_3',h_4'\}$ and h_1 is made adjacent to $p_1^{i,1}$ for all $i \in [n/2]$ and h_1' is made adjacent to $p_4^{i,m(3\frac{n}{2}+1)}$ for all $i \in [n/2]$. This concludes the construction of $G(\sigma)$. See Figure 3.5 for a depiction of the graph G as a matrix of blocks.

Lemma 3.2.2. *If σ has a satisfying assignment, then $G(\sigma)$ has a total dominating set of size at most $4m(3\frac{n}{2}+1)\frac{n}{2}+4$.*

Proof. Let τ be a satisfying truth assignment of σ. Let τ_i be the restriction of τ to the i-th variable pair. We construct a total dominating set X of the desired size as follows. For every variable pair $i \in [\frac{n}{2}]$, we take in each block $B^{i,\ell}$ the vertices in $X^{i,\ell}_{\kappa(\tau_i)}$, the vertex $z^{i,\ell}_{\kappa(\tau_i)}$, and the clique guard $y_1^{i,\ell}$ for all $\ell \in [m(3\frac{n}{2}+1)]$. Additionally, we take the endpoint guards h_1, h_3, h_1', h_3'. This concludes the description of X and we see that $|X| = 4m(3\frac{n}{2}+1)\frac{n}{2}+4$.

First, observe that X dominates all the endpoint guards h_t, h_t', $t \in [4]$, and the endpoints of all P^i are dominated, since $h_1, h_1' \in X$. Next, since X contains two vertices from each path segment $P^{i,\ell}$, we see that X intersects the neighborhoods of all guard vertices in $Q^{i,\ell}$ and hence dominates them. On each path P^i, we are repeatedly taking two consecutive

vertices with a gap of two, except for possibly the ends of P^i. Hence, also all internal vertices of P^i are dominated.

Consider the vertices in the $Z^{i,\ell}$ next. Since $y_1^{i,\ell} \in X$, we see that $y_2^{i,\ell}$ and all $z_j^{i,\ell}$, $j \in [4]$, are dominated. Additionally, since $z_{\kappa(\tau_i)}^{i,\ell} \in X$, also $y_1^{i,\ell}$ and $\hat{z}_{\kappa(\tau_i)}^{i,\ell}$ must be dominated. For $j \in [4] \setminus \{\kappa(\tau_i)\}$, we see that $\hat{z}_j^{i,\ell}$ is dominated by some vertex on $P^{i,\ell}$, because $N(\hat{z}_j^{i,\ell}) \cap X = (P^{i,\ell} \setminus X_j^{i,\ell}) \cap X_{\kappa(\tau_i)}^{i,\ell} \neq \emptyset$ by construction.

Finally, consider the clause vertices $\hat{c}_{\ell'}^{\gamma}$. Since clause $C_{\ell'}$ is satisfied by τ, there is some variable pair i so that τ_i satisfies $C_{\ell'}$. By construction of $G(\sigma)$ and X, we have that $z_{\kappa(\tau_i)}^{i,(\gamma-1)m+\ell'} \in X$ and this vertex is adjacent to $\hat{c}_{\ell'}^{\gamma}$, so $\hat{c}_{\ell'}^{\gamma}$ is dominated. □

Usually, the reverse direction, i.e., showing that a solution to the target problem yields a satisfying assignment, is more difficult, since the transition order comes into play here. We start by studying the structure of total dominating sets in $G(\sigma)$ and showing that they behave as desired, i.e., assume one of the sets $X_j^{i,\ell}$, $j \in [4]$, on each path gadget.

Lemma 3.2.3. *Let X be a total dominating set of $G(\sigma)$, then for any block $B^{i,\ell} = P^{i,\ell} \cup Q^{i,\ell} \cup Z^{i,\ell}$, we have that $|X \cap P^{i,\ell}| \geq 2$, $|X \cap Z^{i,\ell}| \geq 2$, and $y_1^{i,\ell} \in X$. Moreover, if $|X \cap B^{i,\ell}| \leq 4$, then there is a unique $j(i,\ell) \in [4]$ such that $X \cap P^{i,\ell} = X_{j(i,\ell)}^{i,\ell}$ and $X \cap \{z_j^{i,\ell} : j \in [4]\} = \{z_{j(i,\ell)}^{i,\ell}\}$.*

Proof. Suppose that $|X \cap P^{i,\ell}| \leq 1$, then there is some $j \in [4]$ so that $N(q_j^{i,\ell}) \cap X = \emptyset$, hence X cannot be a total dominating set. Observe that $y_2^{i,\ell}$ is a degree-1 vertex, so X must contain its neighbor $y_1^{i,\ell}$. For $y_1^{i,\ell} \in X \cap Z^{i,\ell}$ to be dominated, X must contain another vertex from $N(y_1^{i,\ell}) \subseteq Z^{i,\ell}$, hence $|X \cap Z^{i,\ell}| \geq 2$.

Now, suppose that $|X \cap B^{i,\ell}| \leq 4$ holds. Due to the first part of this lemma, we have that $|X \cap B^{i,\ell}| = 4$, in particular $|X \cap P^{i,\ell}| = 2$, $X \cap Q^{i,\ell} = \emptyset$, and $X \cap \{\hat{z}_j^{i,\ell} : j \in [4]\} = \emptyset$, because $N(y_1^{i,\ell}) \subseteq Z^{i,\ell} \setminus \{\hat{z}_j^{i,\ell} : j \in [4]\}$. Suppose that $X \cap P^{i,\ell} \notin \{X_1^{i,\ell}, \ldots, X_4^{i,\ell}\}$, then $X \cap P^{i,\ell} = \{p_1^{i,\ell}, p_3^{i,\ell}\}$ or $X \cap P^{i,\ell} = \{p_2^{i,\ell}, p_4^{i,\ell}\}$, but then $p_3^{i,\ell}$ or respectively $p_2^{i,\ell}$ is not dominated. Hence, there is a unique $j(i,\ell) \in [4]$ such that $X \cap P^{i,\ell} = X_{j(i,\ell)}^{i,\ell}$. By the established properties of X, the only neighbor of $\hat{z}_{j(i,\ell)}^{i,\ell}$ that may be in X is $z_{j(i,\ell)}^{i,\ell}$, hence to dominate $\hat{z}_{j(i,\ell)}^{i,\ell}$ we must have $z_{j(i,\ell)}^{i,\ell} \in X$. This concludes the proof. □

Lemma 3.2.3 has established that total dominating sets behave on each path gadget as desired. The next step is to show that the transition ordering is respected when sweeping along the path gadgets of a row; recall that $X_1^{i,\ell} = \{p_3^{i,\ell}, p_4^{i,\ell}\}$, $X_2^{i,\ell} = \{p_1^{i,\ell}, p_4^{i,\ell}\}$, $X_3^{i,\ell} = \{p_2^{i,\ell}, p_3^{i,\ell}\}$, $X_4^{i,\ell} = \{p_1^{i,\ell}, p_2^{i,\ell}\}$ and $\mathbf{s}^1 = (\mathbf{0}_0, \mathbf{1}_1), \mathbf{s}^2 = (\mathbf{1}_0, \mathbf{1}_0), \mathbf{s}^3 = (\mathbf{0}_1, \mathbf{0}_1), \mathbf{s}^4 = (\mathbf{1}_1, \mathbf{0}_0)$.

Lemma 3.2.4. *Suppose that X is a total dominating set of $G(\sigma)$ with $|X \cap B^{i,\ell}| \leq 4$ for all i and ℓ. Using the notation from Lemma 3.2.3, it holds for every $i \in [n/2]$ and $j \in [m(3\frac{n}{2}+1)-1]$ that $j(i,\ell) \leq j(i,\ell+1)$.*

Proof. The inequality $j(i,\ell) \leq j(i,\ell+1)$ is trivial if $j(i,\ell+1) = 4$. If $j(i,\ell+1) \leq 2$, then $p_1^{i,\ell+1}$ can only be dominated by $p_4^{i,\ell}$, which implies that $j(i,\ell) \leq 2$. It remains to rule out the cases $j(i,\ell) = 2$, $j(i,\ell+1) = 1$, and $j(i,\ell) = 4$, $j(i,\ell+1) = 3$. In both cases the vertex $p_4^{i,\ell}$ is not dominated. □

We can now prove the reverse direction of the reduction by deducing that a region R^{γ} exists where the state on all paths P^i, $i \in [n/2]$, remains stable and arguing that we can read off a satisfying assignment there.

Lemma 3.2.5. *If $G(\sigma)$ has a total dominating size X of size at most $4m(3\frac{n}{2}+1)\frac{n}{2}+4$, then σ is satisfiable.*

Proof. Note that X contains the vertices h_1, h_3, h'_1, h'_3 due to the degree-1 neighbors. There are in total $m(3\frac{n}{2}+1)\frac{n}{2}$ blocks $B^{i,\ell}$ and X has to contain at least four vertices in each block due to Lemma 3.2.3. This completely uses the budget for X and thus we have $|X \cap B^{i,\ell}| = 4$ for all i and ℓ. By Lemma 3.2.3, X assumes one of our desired states on each $B^{i,\ell}$.

Using the notation of Lemma 3.2.4, we say that a *cheat* occurs if $j(i,\ell) < j(i,\ell+1)$. Due to Lemma 3.2.4, there can be at most 3 cheats on each P^i. Since there are $\frac{n}{2}$ paths, at most $3\frac{n}{2}$ regions can be spoiled by cheats. Hence, there is at least one $\gamma \in [3\frac{n}{2}+1]$ so that R^γ does not contain a cheat. Fix this γ for the remainder of the proof.

We will read off a satisfying truth assignment τ for σ from region R^γ. The truth assignment for variable pair i is given by $\tau_i = \kappa^{-1}(j(i,\gamma m))$; since R^γ is free of cheats, we could have chosen any $\ell \in [(\gamma-1)m+1, \gamma m]$ instead of γm to define τ_i. The desired truth assignment is given by $\tau = \tau_1 \cup \cdots \cup \tau_{n/2}$.

Consider some clause $C_{\ell'}$ and its associated clause vertex $\hat{c}^\gamma_{\ell'}$ in region R^γ. Let $\ell = (\gamma-1)m+\ell'$. Since X dominates the clause vertex, there is some i so that $z^{i,\ell}_{j(i,\ell)} \in X \cap N(\hat{c}^\gamma_{\ell'})$. By construction, this means that the state of X on $P^{i,\ell}$ corresponds to a satisfying truth assignment of clause $C_{\ell'}$. By the previous paragraph, this is the same state as the state of X on $P^{i,\gamma m}$ which in turn corresponds to τ_i, hence τ must satisfy clause $C_{\ell'}$. Since the choice of $C_{\ell'}$ was arbitrary, it follows that σ is satisfiable. □

It remains to bound the cutwidth of the constructed graph $G(\sigma)$ and then we finally have all ingredients to prove Theorem 3.2.1.

Lemma 3.2.6. *The cutwidth of $G(\sigma)$ is at most $n/2 + 4q + \mathcal{O}(1)$.*

Proof. Consider the linear layout of $G(\sigma)$ starting with h_4, h_3, h_2, h_1. Then we go through $G(\sigma)$ column by column and for each column $\ell \in [m(3\frac{n}{2}+1)]$, the linear layout first contains the clause vertex of the ℓ-th column, and then the vertices of $B^{1,\ell}$, $B^{2,\ell}$, …, $B^{n/2,\ell}$ in this order. The layout ends with the vertices h'_1, h'_2, h'_3, h'_4.

Consider any vertex $v^* \in V(G(\sigma))$ and the edges crossing the cut directly after v^*. If $v^* \in \{h_4, h_3, h_2, h'_1, h'_2, h'_3, h'_4\}$, then at most two edges cross the cut. If $v^* = h_1$, then $n/2$ edges cross the cut, namely those from h_1 to the block gadgets $B^{i,1}$ for all $i \in [n/2]$. Now, suppose that v^* is a vertex in column $\ell \in [m(3\frac{n}{2}+1)]$.

If $v^* = \hat{c}^\gamma_{\ell'}$ is the clause vertex in column $\ell = (\gamma-1)m+\ell'$, then the cut consists of the edges between v^* and its neighbors and the $n/2$ matching edges between column $\ell-1$ and column ℓ (or the $n/2$ edges coming from h_1 if $\ell = 1$). Since each clause has size at most q, the clause vertex $\hat{c}^\gamma_{\ell'}$ is adjacent to vertices in at most q different block gadgets. By construction, $v^* = \hat{c}^\gamma_{\ell'}$ is adjacent to at most 4 vertices in each block gadget. Hence, this cut consists of at most $n/2 + 4q$ edges.

If $v^* \neq \hat{c}^\gamma_{\ell'}$ is not the clause vertex in column $\ell = (\gamma-1)m+\ell'$, then v^* lies in some block gadget $B^{i,\ell}$ with $i \in [n/2]$. By construction, every other block gadget is completely contained on one side of the cut, so the cut contains edges that are internal to at most one block gadget. Since each block gadget consists of a constant number of vertices, this accounts for a constant number of edges. Furthermore, the cut contains at most $n/2$ matching edges, one

for each group. More precisely, for the block gadgets of column ℓ that appear before $B^{i,\ell}$ in the layout, the edge from the exit vertex to the entry vertex of the next column is in the cut, and for the block gadgets of column ℓ that appear after $B^{i,\ell}$, the edge to the previous column is in the cut. The block gadget $B^{i,\ell}$ has two edges to other columns that can be in the cut. Finally, the edges from block gadgets of column ℓ to $\hat{c}^{\gamma}_{\ell'}$ can be in the cut, but these are bounded by $4q$ as before. In total, the cut contains at most $n/2 + 4q + \mathcal{O}(1)$ edges. □

As seen in the proof of Lemma 3.2.6, the number of edges incident to $\hat{c}^{\gamma}_{\ell'}$, which depends on the clause size, affects the cutwidth of the construction, hence we assume SETH for this lower bound instead of the weaker hypothesis CNF-SETH.

Proof of Theorem 3.2.1. Suppose there is an algorithm $\mathbb{A}$ that solves Total Dominating Set in time $\mathcal{O}^*((4-\varepsilon)^{\mathrm{ctw}(G)})$ for some $\varepsilon > 0$. We show how to solve q-Satisfiability in time $\mathcal{O}^*((2-\varepsilon')^n)$ for some $\varepsilon' > 0$ for all q, thus contradicting SETH by Theorem 2.1.2. Given a q-Satisfiability instance σ, we construct $G(\sigma)$ in polynomial time and then run $\mathbb{A}$ on $G(\sigma)$ and return its answer. This is correct by Lemma 3.2.2 and Lemma 3.2.5. Due to Lemma 3.2.6, the running time is $\mathcal{O}^*((4-\varepsilon)^{\mathrm{ctw}(G(\sigma))}) \leq \mathcal{O}^*((4-\varepsilon)^{n/2+4q+\mathcal{O}(1)}) \leq \mathcal{O}^*((4-\varepsilon)^{n/2}) \leq \mathcal{O}^*((2-\varepsilon')^n)$ for some $\varepsilon' > 0$, where we use $q \in \mathcal{O}(1)$ in the second inequality. □

3.2.3 Lower Bound for Dominating Set[tc-cutwidth]

The lower bound for Total Dominating Set[cutwidth] can be lifted to a tight lower bound for Dominating Set[tc-cutwidth]. We first survey the relevant known results. Bodlaender et al. [23] have shown that Dominating Set[clique-width] can be solved in time $\mathcal{O}^*(4^{\mathrm{cw}(G)})$. A tight lower bound for Dominating Set[clique-width] was obtained by Katsikarelis et al. [114], as a special case of the more general (k,r)-Center problem. The lower bound of Katsikarelis et al. already applies to the more restrictive parameter linear-clique-width as is usual. As twinclass-cutwidth is even more restrictive than linear-clique-width, cf. Section 2.4.4, our lower bound extends the range of parameters where the optimal base of Dominating Set is 4.

When solving Dominating Set on a graph G containing nontrivial twinclasses, we observe that in true twinclasses C it suffices to pick a single vertex in C which will dominate all vertices inside C and its neighboring twinclasses. However, for false twinclasses C, i.e., those inducing an independent set, we need to take all vertices in C or, preferably, take a single vertex in a neighboring twinclass to dominate all vertices inside C. Hence, if G only consists of false twinclasses, then we essentially need to solve Total Dominating Set on the quotient graph $G/\Pi_{tc}(G)$. By replacing every vertex of a graph with a false twinclass, this idea gives rise to a reduction from Total Dominating Set[cutwidth] to Dominating Set[tc-cutwidth]. We make this reduction formal in the remainder of this subsection.

For the remainder of this subsection assume without loss of generality that G is connected and consists of at least two vertices. We define the graph G' by $V(G') = V(G) \cup \{v' : v \in V(G)\}$ and $E(G') = E(G) \cup \{\{u,v'\},\{u',v\},\{u',v'\} : \{u,v\} \in E(G)\}$, i.e., we obtain G' from G by creating a false twin of each vertex, hence G' has no twinclass of size 1.

Lemma 3.2.7. *It holds that* $\text{tc-ctw}(G') \leq \text{ctw}(G)$.

Proof. We begin by showing that $u \neq v \in V(G)$ are twins in G' if and only if u and v are false twins in G. First, suppose that $\{u, v\} \in E(G)$. By construction of G', this implies that $u' \in N_{G'}(v) \setminus N_{G'}(u)$ and hence u and v cannot be twins in G' in this case. If u and v are false twins in G', then $N_G(u) = N_{G'}(u) \cap V(G) = N_{G'}(v) \cap V(G) = N_G(v)$, so u and v are false twins in G as well. If u and v are false twins in G, then $N_{G'}(u) = N_G(u) \cup \{w' : w \in N_G(u)\} = N_G(v) \cup \{w' : w \in N_G(v)\} = N_{G'}(v)$, so u and v are false twins in G' as well.

By construction of G', the two vertices v and v' are false twins in G' for all $v \in V(G)$. Together with the previous argument and since the twin relation is an equivalence relation, this shows that every twinclass U' of G' has the form $U' = U \cup \{v' : v \in U\}$, where U is a false twinclass of G. For each false twinclass U of G, pick a vertex $v_U \in U$ and consider the map $\Pi_{tc}(G') \ni U' = U \cup \{v' : v \in U\} \mapsto v_U \in U \subseteq V(G)$. If U' and W' are adjacent twinclasses of G', then the vertices v_U and v_W have to be adjacent in G. Hence, we can view $G'/\Pi_{tc}(G')$ as a subgraph of G and see that $\text{tc-ctw}(G') = \text{ctw}(G'/\Pi_{tc}(G')) \leq \text{ctw}(G)$. □

Lemma 3.2.8. *G' has a dominating set of size at most $\bar{b}$ if and only if G has a total dominating set of size at most $\bar{b}$.*

Proof. $\Rightarrow$: Let X' be a dominating set of G'. We want to show that X' can be transformed into a dominating set X of G' with $X \subseteq V(G)$ and $|X| \leq |X'|$, we will then argue that such X must be a total dominating set of G. If $|X' \cap \{v, v'\}| = 1$ for some $v \in V(G)$, then we can assume without loss of generality that $v \in X'$. Suppose that $|X' \cap \{v, v'\}| = 2$ for some $v \in V(G)$. If $X' \cap N_{G'}(v) = X' \cap N_{G'}(v') = \emptyset$, then choose some $w \in N_{G'}(v) \cap V(G) = N_{G'}(v') \cap V(G)$, which exists due to our assumptions about G, and consider $X'' = (X' \setminus \{v'\}) \cup \{w\}$. If instead $X' \cap N_{G'}(v) = X' \cap N_{G'}(v') \neq \emptyset$, then consider $X'' = X' \setminus \{v'\}$. In either case, $|X''| \leq |X'|$ and we claim that X'' is still a dominating set of G'. Only vertices inside $N_{G'}[v']$ may not be dominated anymore. The vertex v' is still dominated, because there is some $w \in X'' \cap N_{G'}(v) = X'' \cap N_{G'}(v')$. The vertices in $N_{G'}(v')$ are still dominated, because $v \in X''$ and $N_{G'}(v) = N_{G'}(v')$. Hence, X'' is a dominating set of G'.

By repeating this argument, we obtain a dominating set X of G' with $|X| \leq |X'|$ and $X \subseteq V(G)$. For every $v \in V(G)$, we have that $v' \notin X$, and hence, there exists some $w \in X \cap N_{G'}(v') = X \cap N_{G'}(v) = X \cap N_G(v)$, where the last equality is due to $X \subseteq V(G)$. This shows that X is a total dominating set of G.

$\Leftarrow$: Let X be a total dominating set of G. We claim that X is also a dominating set of G'. As $G'[V(G)] = G$, any vertex in $V(G)$ is still dominated by X inside G'. Consider some vertex v' for some $v \in V(G)$. Since X is a total dominating set of G, there exists some $w \in X \cap N_G(v) = X \cap N_{G'}(v) = X \cap N_{G'}(v')$ and hence v' is dominated by X as well. So, X must be a dominating set of G'. □

Theorem 3.2.9. *If there is an algorithm $\mathbb{A}$ solving* Dominating Set *in time $\mathcal{O}^*((4-\varepsilon)^{\text{tc-ctw}(G)})$ for some $\varepsilon > 0$, then there is an algorithm $\mathbb{A}'$ solving* Total Dominating Set *in time $\mathcal{O}^*((4-\varepsilon)^{\text{ctw}(G)})$.*

Proof. Suppose $\mathbb{A}$ is such an algorithm. The algorithm $\mathbb{A}'$ first constructs G' in polynomial time, then runs $\mathbb{A}$ on G', which takes time $\mathcal{O}^*((4-\varepsilon)^{\text{tc-ctw}(G')}) \leq \mathcal{O}^*((4-\varepsilon)^{\text{ctw}(G)})$ by

Lemma 3.2.7. Finally, $\mathbb{A}'$ returns the same answer as $\mathbb{A}$. The correctness of $\mathbb{A}'$ follows from Lemma 3.2.8 and the running time of $\mathbb{A}'$ is dominated by running $\mathbb{A}$ on G'. □

Part II

Connectivity Problems on Dense Width Parameters

4 Introduction

Connectivity constraints are a very natural form of global constraints in the realm of graph problems, which are more challenging to handle than local constraints. In this part, we investigate the fine-grained parameterized complexity of several benchmark connectivity problems, such as CONNECTED VERTEX COVER, parameterized by dense width parameters. As mentioned in Section 6.1.1, it was thought for a long time that running times of the form $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ are unavoidable even when parameterized by a sparse width parameter like treewidth. Cygan et al. [49, 51] broke this barrier by introducing the cut-and-count-technique and obtained algorithms with single-exponential running time $\mathcal{O}^*(\alpha^{\mathrm{tw}})$ for many connectivity problems parameterized by treewidth. The cut-and-count-technique reformulates the global connectivity constraint via the counting of consistent cuts, which are locally checkable and thus lead to more straightforward dynamic programming algorithms again. From a fine-grained lens, we are interested in whether the obtained bases α in the running time are optimal. Indeed, for multiple problems parameterized by pathwidth or treewidth this was immediately settled by Cygan et al. [50] by providing appropriate lower bounds under SETH.

A natural follow-up question is how the complexity changes when we move away from treewidth to other parameters. Going to a more restrictive parameter, the base might decrease or stay the same. A recent work by Bojikian et al. [24] gives a comprehensive answer for the parameter cutwidth and indeed both phenomenons occur, e.g., for CONNECTED VERTEX COVER the base decreases from 3 to 2 and for STEINER TREE the base stays at 3. In the other direction, we can move to more expressive parameters, where the base might stay the same, increase, or the problem becomes so difficult that no single-exponential algorithm or even no FPT-algorithm exists anymore. This direction is particularly interesting, as dense graphs cannot have small treewidth and therefore the treewidth results do not give any indication of what the precise complexity of connectivity problems in the dense setting could be. Following this direction, it is natural to consider the parameter *clique-width*, as it is one of the most popular graph parameters to capture dense graphs. The following results on connectivity problems parameterized by clique-width are based on joint work with Stefan Kratsch [100].

4.1 Connectivity Problems Parameterized by Clique-Width

Some results on connectivity problems parameterized by clique-width already exist, but the resulting algorithms are not sufficiently optimized to obtain tight running times. Bergougnoux [9] has applied cut-and-count to several width parameters based on structured neighborhoods with clique-width among these. Moreover, Bergougnoux and Kanté [11], building upon the rank-based approach of Bodlaender et al. [19], obtain single-exponential running times $\mathcal{O}^*(\alpha^{\mathrm{cw}})$ for a large class of connectivity problems parameterized by clique-width. As both articles are aimed at obtaining a breadth of single-exponential algorithms for a large class of problems, the CONNECTED (CO-)(σ,ρ)-DOMINATING SET problems, the obtained bases for particular problems are far from being optimal. For example, the former article implies an $\mathcal{O}^*(128^{\mathrm{cw}})$-time algorithm for CONNECTED DOMINATING SET and the latter article yields an $\mathcal{O}^*((27\cdot 2^{\omega+1})^{\mathrm{cw}})$-time algorithm for CONNECTED VERTEX COVER and an $\mathcal{O}^*((8\cdot 2^{\omega+1})^{\mathrm{cw}})$-time algorithm for CONNECTED DOMINATING SET, where ω is the matrix multiplication exponent, see e.g. Alman and Vassilevska W. [3]. Even if $\omega=2$, this only yields the very large bases 216 and 64 respectively.

We show that the running times for CONNECTED VERTEX COVER[clique-width] and CONNECTED DOMINATING SET[clique-width] can be considerably improved by developing faster algorithms relying on the cut-and-count-technique. The faster running times are due to using the cut-and-count-technique, whose running times compare favorably to the rank-based approach already when parameterizing by treewidth, and since the algorithms are fine-tuned by precisely analyzing which cut-and-count states are necessary to consider. This fine-tuning requires techniques such as fast subset convolution, inclusion-exclusion states, and distinguishing between live and dead labels to obtain the improved running times.

Theorem 4.1.1. *There are one-sided-error Monte-Carlo algorithms that, given a k-expression for a graph G, can solve*

- CONNECTED VERTEX COVER *in time* $\mathcal{O}^*(6^k)$,
- CONNECTED DOMINATING SET *in time* $\mathcal{O}^*(5^k)$.

We show that these running times are essentially optimal by proving appropriate lower bounds under SETH. These lower bounds are proved by following the by-now standard construction principle of Lokshtanov et al. [126] for lower bounds relative to width parameters. To apply this principle for clique-width, we closely investigate the problem behavior across *joins*, i.e., the edge-structures via which clique-width is defined, and the results of this investigation strongly guide us in designing appropriate gadgets. Precisely, we obtain the following tight lower bounds:

Theorem 4.1.2. *Assuming SETH, the following statements hold for all $\varepsilon>0$:*

- CONNECTED VERTEX COVER *cannot be solved in time* $\mathcal{O}^*((6-\varepsilon)^{\mathrm{cw}})$.
- CONNECTED DOMINATING SET *cannot be solved in time* $\mathcal{O}^*((5-\varepsilon)^{\mathrm{cw}})$.

The techniques behind the algorithms and lower bounds can also be applied to other connectivity problems, but do not necessarily lead to tight results. Using the same approach, we

provide additional algorithms for Steiner Tree, Connected Odd Cycle Transversal, and a generalization of Connected Vertex Cover and Connected Odd Cycle Transversal to more colors.

Theorem 4.1.3. *There are one-sided-error Monte-Carlo algorithms that, given a k-expression for a graph G, can solve*

- Steiner Tree *in time* $\mathcal{O}^*(4^k)$,
- Connected Odd Cycle Transversal *in time* $\mathcal{O}^*(14^k)$,
- *for every* $q \geq 2$, Connected Deletion to q-Colorable *in time* $\mathcal{O}^*((2^{q+2}-2)^k)$.

4.2 Connectivity Problems Parameterized by Modular-Treewidth

Beyond the results for clique-width, we consider the parameters modular-pathwidth and modular-treewidth, which can be seen as natural intermediate steps between pathwidth/treewidth and clique-width[1]. While modular-treewidth is more restrictive than clique-width, we have much better algorithms for computing modular-treewidth than for clique-width, as the algorithms for treewidth transfer to modular-treewidth, see Theorem 2.4.8.

We obtain the first tight running times for connectivity problems parameterized by modular-treewidth; these results are based on joint work with Stefan Kratsch [101]. These results are achieved by the same general approach as for clique-width, where the finely-tuned algorithms rely on the cut-and-count-technique and the lower bounds use the construction principle of Lokshtanov et al. [126] together with a precise analysis of the problem behavior on the characteristic separators for modular-treewidth. The used techniques seem better suited to modular-treewidth, as they allow us to obtain tight results for a broader range of problems compared to clique-width; in addition to Connected Vertex Cover and Connected Dominating Set, we also obtain tight results for Steiner Tree and Feedback Vertex Set parameterized by modular-treewidth.

Whereas for both considered problems parameterized by clique-width the running time strictly increases compared to treewidth, for modular-treewidth we will also see examples where the complexity remains the same. Indeed, a central observation is that all vertices inside a module must be connected already when a single vertex from a neighboring module is chosen. In the case of Steiner Tree and Connected Dominating Set, this observation yields a reduction from the modular-treewidth case to treewidth, showing that we can deal with a greater generality in the input structure at no cost in complexity for these problems.

Theorem 4.2.1. *There are one-sided error Monte-Carlo algorithms that, given a tree decomposition of width k for every[2] prime quotient graph $G_M^q \in \mathcal{H}_p(G)$, can solve*

- Steiner Tree *in time* $\mathcal{O}^*(3^k)$,
- Connected Dominating Set *in time* $\mathcal{O}^*(4^k)$.

[1] For modular-pathwidth, we have $\mathrm{cw} \preceq \text{mod-pw} \preceq \mathrm{pw}$. However, for modular-treewidth such a relationship cannot hold, as Corneil and Rotics [40] show that for every k there exists a graph G_k with treewidth k and clique-width exponential in k. We also refer to Section 2.4.4

[2] For both problems, we actually only need a tree decomposition of a single quotient graph, however which quotient graph is appropriate depends on the problem instance.

These bases are already optimal under SETH for parameterization by pathwidth by known results of Cygan et al. [50]. Due to $\text{mod-tw} \preceq \text{tw} \preceq \text{pw}$, cf. Lemma 2.4.22, the bases must be tight for modular-treewidth as well and we do not need to provide any new lower bounds.

In contrast, for the problems Connected Vertex Cover and Feedback Vertex Set, the interaction of the problem constraints with the modular structure yields additional complications that are not present in the treewidth case. For both problems, we design new cut-and-count algorithms solving a more involved variant of the problem by dynamic programming along the tree decompositions of the quotient graphs. For Connected Vertex Cover the simpler behavior of the connectivity constraint relative to modules allows us to save one state compared to clique-width. For Feedback Vertex Set[clique-width], it is even unclear how to obtain any reasonable algorithm using cut-and-count at all, since the standard cut-and-count approach for Feedback Vertex Set only yields an XP-algorithm as already observed by Bergougnoux and Kanté [11]. However, in the case of modular-treewidth, the techniques for treewidth still transfer and we obtain an optimized single-exponential algorithm with considerable technical effort.

Theorem 4.2.2. *There are one-sided error Monte-Carlo algorithms that, given a tree decomposition of width k for every prime quotient graph $G_M^q \in \mathcal{H}_p(G)$, can solve*

- Connected Vertex Cover *in time* $\mathcal{O}^*(5^k)$,
- Feedback Vertex Set *in time* $\mathcal{O}^*(5^k)$.

We complement the algorithms for Connected Vertex Cover and Feedback Vertex Set by providing new lower bounds which show that the obtained running times are optimal under SETH. A notable feature of these lower bounds is that they also apply for the more restrictive parameter *twinclass-pathwidth*[3], so the *recursive* partitioning allowed by the modular decomposition is not required, instead a single level of partitioning already brings forth sufficient complexity. Compared to clique-width, while the lower bound proofs do follow the same general structure, the restrictiveness of the parameter twinclass-pathwidth makes the concrete gadget design more challenging.

Theorem 4.2.3. *Unless SETH fails, the following statements hold for any $\varepsilon > 0$:*

- Connected Vertex Cover *cannot be solved in time* $\mathcal{O}^*((5-\varepsilon)^{\text{tc-pw}})$.
- Feedback Vertex Set *cannot be solved in time* $\mathcal{O}^*((5-\varepsilon)^{\text{tc-pw}})$.

4.3 Fine-Grained Complexity Landscape

We summarize the main results obtained in this part and compare them with the tight bounds obtained for cutwidth by Bojikian et al. [24] and treewidth by Cygan et al. [51] in Table 4.1. The table precisely quantifies the *price of generality* for the considered connectivity problems when moving from restrictive parameters to more expressive parameters.

All algorithms considered in the table use the cut-and-count-technique in some fashion and are therefore randomized. For Feedback Vertex Set, the cut-and-count-technique

[3] By Corollary 2.4.27 and Lemma 2.4.22, we have $\text{cw} \preceq \text{mod-pw} \preceq \text{tc-pw}$.

Parameters	cutwidth	treewidth	modular-tw	clique-width
Connected Vertex Cover	2^k	3^k	5^k	6^k
Connected Dominating Set	3^k	4^k	4^k	5^k
Steiner Tree	3^k	3^k	3^k	between 3^k and 4^k
Feedback Vertex Set	2^k	3^k	5^k	at least 5^k
References	[24]	[50, 51]	here ([101])	here ([100])

Tab. 4.1.: Optimal functions $f(k)$ in the running time $\mathcal{O}^*(f(k))$ of several connectivity problems with respect to various width parameters listed in increasing generality. The results in the last two columns are obtained in this thesis. Between modular-treewidth and clique-width, we only have the relationship $\mathrm{cw}(G) \leq 3 \cdot 2^{\text{mod-tw}(G)-1}$, but the same results are also tight for the more restrictive modular-pathwidth, where we have $\mathrm{cw} \preceq \text{mod-pw}$ by Corollary 2.4.27. For Feedback Vertex Set[clique-width] a single-exponential algorithm is known by Bergougnoux and Kanté [11], but the obtained base in the running time is at least 120.

is applied in all cases by using the fact that a graph G with n vertices and m edges is a forest if and only if G has at most $n - m$ connected components. The other considered treewidth-algorithms are straightforward applications of cut-and-count, where for Connected Dominating Set a fast subset convolution algorithm is used to keep the running time small. The improved cutwidth-algorithms combine cut-and-count either with a generic application of rank-based-methods for cutwidth that avoid Gaussian Elimination, which were introduced by Groenland et al. [95], or for Feedback Vertex Set by essentially removing one of the states from the treewidth-algorithm by using a special edge-subdivision trick.

All lower bounds use the construction principle of Lokshtanov et al. [126] and the creation of a *root vertex*, or *root path* in the case of cutwidth, that allows the gadgets to assume additional states which are crucial to obtain sufficiently strong lower bounds. The complexity of the constructed gadgets roughly increases with the base of the running time, as the base directly corresponds to how many states the so-called *path gadgets* must be able to attain, which necessitates more expressive and varied gadgets for larger bases. A noteworthy feature of the lower bounds for the dense width parameters is that we must select appropriate states from a possibly much larger set of separator states, as not all such choices lead to a transition ordering. We discuss this feature in much more detail when developing the concrete lower bounds.

We remark upon two interesting phenomenons in Table 4.1. First, when comparing the rows for Connected Vertex Cover and Connected Dominating Set, we see that while Connected Vertex Cover starts at a lower complexity for the sparse parameters cutwidth and treewidth, the complexity of Connected Vertex Cover overtakes Connected Dominating Set when going to the dense width parameters. An intuitive explanation for this phenomenon is that the considered states for the dense parameters describe the behavior of *groups* of vertices and that connectivity constraints often supersede domination constraints in this setting: both unconnected vertices and undominated vertices in a group require that some vertex in an adjacent group is put into the solution. For Connected Vertex Cover, the problem constraints and connectivity do not coalesce in this way and hence keep contributing to different states. Secondly, we compare with the problem variants without connectivity constraints, i.e., Vertex Cover and Dominating Set. The optimal base for Vertex Cover is 2 for all considered parameters by results of van Geffen et al. [94], Courcelle et al. [43], and the modular-treewidth algorithm in this thesis, cf. Theorem 6.2.2.

The optimal bases for DOMINATING SET are 3 for cutwidth and treewidth by van Geffen et al. [94] and van Rooij et al. [164], and 4 for modular-pathwidth and clique-width by our lower bound, cf. Section 3.2.3, and Bodlaender et al. [23][4]. This shows that the impact of introducing a connectivity constraint on the problem complexity greatly depends on the considered problem and the considered parameter: for DOMINATING SET the complexity increases by at most 1 in all considered cases and for VERTEX COVER the complexity increase ranges between 0 and 4 going from the restrictive cutwidth to the expressive clique-width. In particular, the phenomenon observed by Cygan et al. [51] that imposing a connectivity constraint increases the complexity by at most 1 in the sparse setting does not extend to dense width parameters.

4.4 Related Work

HAMILTONIAN CYCLE is a connectivity problem that we have not studied here, but for which several tight results are known. Cygan et al. [51] show with the cut-and-count-technique that HAMILTONIAN CYCLE[treewidth] can be solved in time $\mathcal{O}^*(4^k)$, it is however unknown whether this running time is optimal under SETH. Cygan, Kratsch, and Nederlof [48] refine the rank-based approach for HAMILTONIAN CYCLE[pathwidth], in particular avoiding the expensive basis computation using additional tricks, and obtain an $\mathcal{O}^*((2+\sqrt{2})^k)$-time algorithm; surprisingly, they also show that this base is optimal under SETH. Bodlaender et al. [19] show using the squared determinant approach that Hamiltonian cycles can be counted in time $\mathcal{O}^*(6^k)$ parameterized by pathwidth. Curticapean et al. [45] show that this running time is tight under SETH and also consider counting modulo a prime: the algorithm of Cygan, Kratsch, and Nederlof [48] also solves the counting modulo 2 variant and Curticapean et al. [45] show for all other primes that the running time base must be at least $2 + 1.97 > 2 + \sqrt{2}$ when parameterizing by pathwidth. By improving the convolution algorithm used at the join node of a tree decomposition, Wlodarczyk [174] shows that Hamiltonian cycles can also be counted in time $\mathcal{O}^*((2^\omega+2)^k)$ when parameterized by treewidth, which matches the running time of the pathwidth-algorithm if the matrix multiplication exponent satisfies $\omega = 2$. As an edge selection problem, HAMILTONIAN CYCLE becomes much more difficult when considering dense width parameters; indeed, Fomin et al. [73] show that HAMILTONIAN CYCLE[clique-width] is W[1]-hard, thus ruling out any FPT-algorithm under the common assumption FPT $\neq$ W[1]. Bergougnoux et al. [13] provide an XP-algorithm for HAMILTONIAN CYCLE[clique-width] whose running time is tight under ETH.

Beyond the already mentioned results and the discussion in Section 3.1.3, the cut-and-count-technique has also been used with the parameter branchwidth by Pino et al. [156]; it is unknown whether their results are tight and the obtained running times depend on the matrix multiplication exponent ω. Groenland et al. [95] use a variant of the cut-and-count-technique to obtain a tight algorithm for counting the number of connected spanning edge

[4] As we only have $\mathrm{cw} \preceq \mathrm{mod\text{-}pw}$ and *not* $\mathrm{cw} \preceq \mathrm{mod\text{-}tw}$, the algorithm for clique-width by Bodlaender et al. only carries over to modular-pathwidth and not to modular-treewidth. We believe that an $\mathcal{O}^*(4^k)$-time algorithm for DOMINATING SET[modular-treewidth] should also follow by the techniques in this thesis, but it is technically not written down.

sets modulo a prime parameterized by cutwidth. Furthermore, there are applications of the cut-and-count-technique relative to the parameter treedepth, but these will be discussed in more detail in Part III.

In addition to the application of the cut-and-count-technique on graphs with structured neighborhoods by Bergougnoux [9, Section 4.3], Bergougnoux and Kanté [12] also apply the rank-based approach to connectivity problems on such graphs. Both articles capture graphs with structured neighborhoods via the notion of d-*neighbor equivalence* introduced by Bui-Xuan et al. [31]; two subsets X, Y of a fixed set $A \subseteq V(G)$ are d-*neighbor equivalent* if every vertex in $V(G) \setminus A$ has the same number of neighbors in X and Y or at least d neighbors in both sets. Informally, a graph has structured neighborhoods if it can be decomposed by only considering sets $A \subseteq V(G)$ such that the d-neighbor relation relative to A has few equivalence classes. The width parameters clique-width, rank-width, and mim-width (and more restrictive parameters) can be captured in this way. Therefore, their results yield algorithms for a broad range of problems and parameters, but are consequently not very optimized (in the SETH sense) for specific problem-parameter-combinations as already discussed. Building upon these methods, Bergougnoux et al. [10] consider a special logic that can handle connectivity constraints and they obtain a model-checking algorithm for this logic. This model-checking algorithm yields XP running times for parameterization by mim-width and single-exponential running times for treewidth and clique-width; the resulting running times are tight under ETH for many cases.

Moving away from connectivity problems, we survey some more of the literature obtaining tight fine-grained parameterized algorithms for dense parameters. Iwata and Yoshida show that for any $\varepsilon > 0$ Vertex Cover[treewidth] can be solved in time $\mathcal{O}^*((2-\varepsilon)^k)$ if and only if Vertex Cover[clique-width] can be solved in time $\mathcal{O}^*((2-\varepsilon)^k)$ [106]; as the bases differ for treewidth and clique-width in our case, it seems difficult to transfer their techniques to our setting. Lampis [123] obtains the tight running time of $\mathcal{O}^*((2^q-2)^k)$ for q-Coloring[clique-width] and a tight result for q-Coloring[twinclass-treewidth]. Generalizing to homomorphism problems, Ganian et al. [86] obtain tight results for parameterization by clique-width, where the obtained base depends on a special measure of the target graph. Katsikarelis et al. [114] obtain tight results for (t,r)-Center[clique-width] and, in particular, the tight running time $\mathcal{O}^*(4^{\mathrm{cw}})$ for Dominating Set. Jacob et al. [108] have, simultaneously with us, shown that the running time $\mathcal{O}^*(4^k)$ is tight for Odd Cycle Transversal[clique-width], however our lower bound applies to a parameterization more restrictive than twinclass-pathwidth and considers a generalization to more colors, see Chapter 13.

4.5 Organization

We first present the algorithmic results of this part. Chapter 5 contains the algorithms parameterized by clique-width and Chapter 6 contains the algorithms parameterized by modular-treewidth and the reductions to the treewidth case; both of these sections start with an overview of dynamic programming for the respective parameter and a development of appropriate tools, afterwards we go into the details of the concrete algorithms. Then, we move on to the lower bounds, where Chapter 7 contains the lower bounds parameterized by

clique-width and Chapter 8 contains the lower bounds parameterized by modular-treewidth. Each lower bound section begins with how the general lower bound principle, which we presented in Section 3.2.1, applies to the considered parameter and we then proceed to the concrete lower bounds. Every lower bound proof begins with an intuitive analysis of the problem structure with respect to the parameter-dependent separators based on compatibility matrices, which strongly guides the design process of the gadgets. We conclude this part in Chapter 9 and present several avenues for further research together with possible approaches in many cases.

5 Algorithms Parameterized By Clique-Width

5.1 Dynamic Programming on Clique Expressions

We begin by giving the basic definitions and ideas for dynamic programming on clique-expressions.

Basic Definitions. Let μ be a k-expression for $G = (V, E)$; the associated syntax tree is T_μ. Recall that for $t \in V(T_\mu)$, we denote by μ_t the subexpression induced by the subtree of T_μ rooted at t. Whenever a clique-expression μ is fixed, we define $G_t = G_{\mu_t}$, $V_t = V(G_t)$, $E_t = E(G_t)$, and $\mathtt{lab}_t = \mathtt{lab}_{\mu_t}$ for any $v \in V(T_\mu)$. Furthermore, we write $V_t^\ell = \mathtt{lab}_t^{-1}(\ell)$ for the set of all vertices with label ℓ at node t and we write $L_t = \{\ell \in \mathbb{N} : V_t^\ell \neq \emptyset\}$ for the set of *nonempty labels at node* t.

Basic Dynamic Programming. To solve a graph problem given a clique-expression μ with k labels, we first define a family $\mathcal{A}_t$ of sensible partial solutions associated with the subgraph G_t; the family $\mathcal{A}_{\hat{r}}$ at the root-node $\hat{r}$ of T_μ should essentially contain all solutions to the problem. To compute these families, at least implicitly, we want to move bottom-up along the syntax tree T_μ, computing $\mathcal{A}_t$ via recurrences based on the type of the operation associated with t which combine or update the partial solution families at the child node(s) of t. However, to formulate such recurrences, we need information on how the partial solutions interact with the *label classes* V_t^ℓ, also just referred to as *labels*. These interactions, also called *states*, depend on the considered partial solution and we capture them by *t-signatures* which are functions of the form $f\colon L_t \to \mathbf{States}$, where $\mathbf{States}$ is a problem-dependent set, mapping each label to its state. Thus, we obtain subfamilies $\mathcal{A}_t(f) \subseteq \mathcal{A}_t$ containing all partial solutions at G_t that are compatible with the t-signature f. The desired recurrences are then formulated by taking the t-signatures f into account. We will not directly compute $\mathcal{A}_t(f)$, but a statistic associated with it, such as its cardinality or the minimum cost of a contained partial solution; this statistic is usually denoted by $A_t(f)$.

Number of Nodes. Courcelle and Olariu [44] show that we can assume that the associated syntax tree T_μ contains at most $\mathcal{O}(k^2 n) = \mathcal{O}^*(1)$ nodes. Via local swapping transformations, one can assume that between any two union-operations at most $\mathcal{O}(k^2)$ join-operations and at most $\mathcal{O}(k)$ relabel operations occur. Since any clique-expression for G contains n introduce-operations and $n-1$ union-operations, the result follows.

5.1.1 Algorithmic Techniques

We survey some of the additional algorithmic techniques used to obtain the algorithms for Connected Vertex Cover[clique-width] and Connected Dominating Set[clique-width].

Lifting Vertex States to Label States. For dynamic programming along clique-expressions, we have to characterize the relevant interactions of a partial solution with the *labels* which govern which *joins* can be constructed by the expression. In the considered problems, a single vertex v can take a constant number of different states with respect to a partial solution which we capture with a problem-dependent set Ω; e.g., for Connected Vertex Cover, we have $\Omega = \{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$, representing $v \notin X$ (state $\mathbf{0}$), $v \in X_L$ (state $\mathbf{1}_L$), and $v \in X_R$ (state $\mathbf{1}_R$), respectively. A clique-expression repeatedly adds *joins* between pairs of vertex sets, say A and B, i.e., all possible edges between A and B are added, and the algorithm must check whether a partial solution remains feasible after adding a join and possibly update some states. A priori, each choice of vertex states in a label could yield different behaviors for partial solutions. However, the crucial observation for the considered problems is that the precise multiplicity of a vertex state in A or in B is irrelevant for a join, rather it suffices to distinguish which vertex states appear on each side and which do not. Therefore, the relevant label states are captured by the subsets of Ω. The next two techniques will allow us to reduce the number of considered states further.

Nice Clique-Expressions. For both algorithms, we refine and augment standard clique-expressions to distinguish between *live* and *dead* labels. When performing dynamic programming along a clique-expression, we consider the induced subgraphs defined by subexpressions of the given clique-expression. At a subexpression, we say that a label ℓ is *live* if in the remaining expression the vertices with label ℓ receive further edges that are not present in the current subexpression, otherwise we say that ℓ is *dead*. First, we observe that we do not need to track the states of a partial solution at the dead labels, as they only have trivial interactions with the other states in the remaining expression. Hence, we only need to consider the states that can be attained at live labels which allows us to reduce the number of considered states for Connected Vertex Cover. To simplify the description of the algorithms and avoid handling edge cases, we transform the clique-expressions so that no degenerate cases occur and add a dead-operation $\perp_\ell$ which handles label ℓ turning from live to dead. The dead-operation is similar to *forget vertex nodes* in *nice tree decompositions* [116] and to the delete-operation in multi-clique-expression [79]. Distinguishing live and dead labels has been used before [86, 114, 123] to obtain improved running times, but handling the label types explicitly via an additional operation is new to the best of the author's knowledge. We formally develop the nice clique-expressions in Section 5.1.2.

Inclusion-Exclusion States. For Connected Dominating Set, we transform to a different set of vertex states, called *inclusion-exclusion states*, which have proven helpful for domination problems before [99, 140, 155, 164]. With these states we do not track whether a vertex is *undominated* or *dominated* by a partial solution, but rather *allow* a vertex to be dominated or *forbid* it. A solution to the original problem can usually be recovered by

an inclusion-exclusion argument, however when lifting to label states this argument does not directly transfer. We show that the argument can be adapted for the label states when working modulo two, whereas for vertex states the argument is known to also work for non-modular counting. The advantage of the inclusion-exclusion states is that at join-operations we do not have to update vertex states from undominated to dominated, thus simplifying the algorithm and also allowing us to collapse several label states into a single one. The dead-operations of nice clique-expressions serve as suitable time points in the algorithm to apply the adapted inclusion-exclusion argument.

Fast Convolutions. To quickly compute the dynamic programming recurrences, we utilize algorithms for *fast subset convolution*. In Section 5.2, we tailor the techniques developed by Björklund et al. [15] on trimmed subset convolutions to obtain a fast algorithm for the union-recurrence appearing in the Connected Vertex Cover algorithm. For Connected Dominating Set, the lattice-based results of Björklund et al. [16] provide the necessary means to compute the union-recurrence quickly. In both cases, we obtain fast convolution algorithms applicable in more general settings, so these results could be of independent interest.

5.1.2 Nice Clique-Expressions

Irredundancy. A clique-expression μ is *irredundant* if for any join-operation $\eta_{i,j}(G_{t'}) = t \in V(T_\mu)$, it holds that $E_{G_{t'}}(V_{t'}^i, V_{t'}^j) = \emptyset$, i.e., no edge added by the join existed before.

Theorem 5.1.1 ([44]). *Any k-expression μ can be transformed into an equivalent, i.e., $G_{\mu'} = G_\mu$, irredundant k-expression μ' in polynomial time.*

Lemma 5.1.2 ([13]). *If μ is an irredundant k-expression for the graph $G = (V, E)$ and $t \in V(T_\mu)$, then for all labels $\ell \in L_t$ and vertices $u, v \in V_t^\ell$ we have that $N_G(u) \setminus V_t = N_G(v) \setminus V_t$. Furthermore, if $u \in V_t^i$, $v \in V_t^j$ with $i \neq j \in L_t$ and $\{u, v\} \in E \setminus E_t$, then $E_G(V_t^i, V_t^j) \subseteq E \setminus E_t$.*

Henceforth, we will assume that the given k-expression μ is irredundant. Irredundancy still allows several edge cases regarding empty labels to occur, which would require special handling in the dynamic programming algorithms. To avoid this extra effort in the algorithms, we show how to transform any clique-expression so that these edge cases do not occur.

Definition 5.1.3. We say that a clique-expression μ of a graph $G = G_\mu$ is *nice* if μ satisfies the following properties:

- μ is irredundant,
- for every join-node $\eta_{i,j}(G_{t'}) = t \in V(T_\mu)$, where t' is the child of t, we have that $G_t \neq G_{t'}$, i.e., t adds at least one edge and $V_{t'}^i \neq \emptyset$ and $V_{t'}^j \neq \emptyset$,
- for every relabel-node $\rho_{i \to j}(G_{t'}) = t \in V(T_\mu)$, where t' is the child of t, we have that $V_{t'}^i \neq \emptyset$ and $V_{t'}^j \neq \emptyset$.

The following lemma shows that preprocessing allows us to assume that we are given a nice k-expression. A more involved proof that this can even be done in linear time was also implicitly obtained by Ducoffe in the proof of Lemma 7 in [61].

Lemma 5.1.4 ([61]). *Any k-expression μ can be transformed into an equivalent, i.e., $G_{\mu'} = G_\mu$, nice k-expression μ' in polynomial time.*

Proof. First running the algorithm of Theorem 5.1.1, we can assume that μ is already irredundant. If a join node $t = \eta_{i,j}(G_{t'})$ does not add any edges, then we must have $V^i_{t'} = \emptyset$ or $V^j_{t'} = \emptyset$ by irredundancy. Clearly, we can simply remove such join-nodes from the expression. The next step is to observe that any relabel node of the form $\rho_{i\to j}(G_{t'}) = t \in V(T_\mu)$, where t' is the child of t and $V^i_{t'} = \emptyset$, can be removed from μ without changing the resulting graph, as no label is changed by such a node.

Now, suppose that μ contains $p > 0$ relabel nodes of the form $\rho_{i\to j}(G_{t'}) = t \in V(T_\mu)$, where t' is the child of t and $V^i_{t'} \neq \emptyset$ and $V^j_{t'} = \emptyset$; we call such relabels *unnecessary*. We pick one such occurrence and show how to remove it. If t is the root of μ, then we can simply remove t without changing the resulting graph. If t is not the root of μ, then let s be the parent node of t. We swap the role of label i and j in every proper descendant of t in T_μ and remove the node t letting t' be a child of s in place of t. The correctness of the transformation can be proved by straightforward bottom-up induction along T_μ and this transformation does not create any new unnecessary relabels.

By repeating this transformation for every unnecessary relabel, we obtain an equivalent nice k-expression μ' in polynomial time. □

Live and Dead Labels. When designing tight algorithms for problems parameterized by clique-width, one often observes that there are states that cannot be attained by a label unless the vertices with this label have already received all their incident edges by the current subexpression. But for such labels, the dynamic programming algorithm does not need to store the state as there will be no interesting interaction with other labels in the remaining expression. So, to improve the running time, we only store the state for *live* labels, i.e., labels that contain vertices that are still missing some incident edges; labels that are not live are called *dead*. This idea has been used several times before for dynamic programming on clique-expressions [86, 114, 123].

While one could precompute for a given clique-expression μ which labels at a node $t \in V(T_\mu)$ are live, we choose to explicitly mark when a label is no longer live in the syntax tree. To do so, we augment the syntax tree with *dead nodes* after join nodes that change at least one label from live to dead. The function of these dead nodes is comparable to that of forget vertex nodes in a nice tree decomposition [116]. Especially in the algorithm for Connected Dominating Set, this explicitness helps because it allows us to cleanly separate two computations, namely the standard computation for join nodes and an extra computation that has to be performed when a label turns dead. We now proceed with the formal definitions.

For the remainder of this section, we assume that G is a connected graph with at least two vertices.

Definition 5.1.5. Given a clique-expression μ for $G = (V, E)$ and a node $t \in V(T_\mu)$, the set of *dead vertices at* t is defined by $D_t = \{v \in V_t : \delta_G(v) \subseteq E_t\}$. A vertex $v \in V_t \setminus D_t$ is called *live at* t.

Lemma 5.1.6. *Given an irredundant k-expression μ for $G = (V, E)$, a node $t \in V(T_\mu)$, and a nonempty label $\ell \in L_t$, we have that either $V_t^\ell \cap D_t = \emptyset$ or $V_t^\ell \subseteq D_t$.*

Proof. If $x \in V_t^\ell \setminus D_t \neq \emptyset$, then there exists an edge $\{x, y\} \in E \setminus E_t$. Let $\tilde{t} \in V(T_\mu)$ be the lowest ancestor of t such that $x, y \in V_{\tilde{t}}$; we either have $\tilde{t} = t$ or $\tilde{t}$ is some union node above t. In either case, we can assume that there are $i \neq j \in L_{\tilde{t}}$ with $x \in V_t^\ell \subseteq V_{\tilde{t}}^i$ and $y \in V_{\tilde{t}}^j$. By the second part of Lemma 5.1.2, we see that $\{x', y\} \in E \setminus E_{\tilde{t}} \subseteq E \setminus E_t$ for all $x' \in V_t^\ell \subseteq V_{\tilde{t}}^i$. Hence, $V_t^\ell \setminus D_t \neq \emptyset$ implies that $V_t^\ell \cap D_t = \emptyset$ which proves the lemma. □

If we do not require irredundancy, then it is easy to construct clique-expressions where Lemma 5.1.6 fails, i.e., $\emptyset \neq V_t^\ell \cap D_t \neq V_t^\ell$. By considering only irredundant clique-expressions, we can say that a whole label is dead or live which simplifies the handling of dead vertices, as we can perform a single computation once a label turns dead. In particular, without irredundancy a label containing only dead vertices at one node might get new live vertices later on in the expression and it is often unclear how to handle such cases.

The following definition formalizes the handling of live and dead labels and the dead nodes that are added when a label turns from live to dead.

Definition 5.1.7. Given an irredundant k-expression for $G = (V, E)$, the *augmented syntax tree* $\hat{T}_\mu$ of μ is obtained from the syntax tree T_μ by inserting up to two *dead nodes* directly above every join node $t = \eta_{i,j}(G_{t'})$, where t' is the child of t in T_μ, based on the following criteria:

- if $V_t^i \subseteq D_t \setminus D_{t'}$, then the node $\bot_i$ is inserted,
- if $V_t^j \subseteq D_t \setminus D_{t'}$, then the node $\bot_j$ is inserted,
- if both nodes $\bot_i$ and $\bot_j$ are inserted, then we insert them in any order.

We extend the notations G_t, V_t, D_t, V_t^ℓ, for $\ell \in [k]$, to dead nodes by inheriting the values of the child node.

For every node $t \in V(\hat{T}_\mu)$ of the augmented syntax tree, we inductively define the set of *live labels* $L_t^{live} \subseteq L_t$ by

$$L_t^{live} = \begin{cases} \{\ell\} & \text{if } t = \ell(v), \\ L_{t'}^{live} \setminus \{i\} & \text{if } t = \rho_{i \to j}(G_{t'}), \\ L_{t'}^{live} & \text{if } t = \eta_{i,j}(G_{t'}), \\ L_{t'}^{live} \setminus \{\ell\} & \text{if } t = \bot_\ell(G_{t'}), \\ L_{t_1}^{live} \cup L_{t_2}^{live} & \text{if } t = G_{t_1} \oplus G_{t_2}. \end{cases}$$

Dually, the set of *dead labels* $L_t^{dead} \subseteq L_t$ is given by $L_t^{dead} = L_t \setminus L_t^{live}$.

See Figure 5.1 for an example of an augmented syntax tree. We now show that, up to pending dead nodes, L_t^{live} contains all nonempty labels that only consist of live vertices at t. Due to Lemma 5.1.6, no label of an irredundant k-expression can contain both live and dead vertices simultaneously.

Lemma 5.1.8. *Let μ be a nice k-expression of $G = (V, E)$ and $\hat{T}_\mu$ its augmented syntax tree. For all $t \in V(\hat{T}_\mu)$ and $\ell \in L_t$, we have that $V_t^\ell \cap D_t = \emptyset$ implies $\ell \in L_t^{live}$. If t is not the child of a dead node, then we even have for every $\ell \in L_t$ that $V_t^\ell \cap D_t = \emptyset$ if and only if $\ell \in L_t^{live}$.*

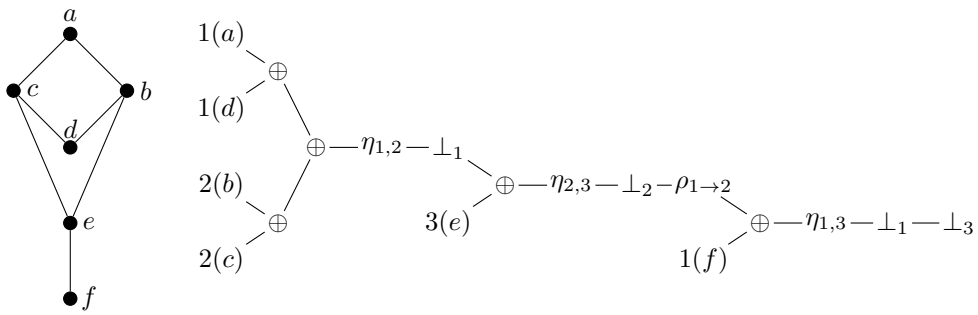


Fig. 5.1.: The left side depicts a graph G and the right side depicts the augmented syntax tree of a nice 3-expression for G.

Proof. First, recall that $V_t^\ell \cap D_t \in \{\emptyset, V_t^\ell\}$ for every $t \in V(\hat{T}_\mu)$ and $\ell \in L_t$ by Lemma 5.1.6. We prove the statement inductively along the augmented syntax tree $\hat{T}_\mu$ by making a case distinction based on the current node type. Note that only join nodes and other dead nodes can be children of a dead node.

- If $t = \ell(v)$, then $D_t = \emptyset$ as v cannot be isolated by assumption and the statement is trivially true.
- If $t = \rho_{i\to j}(G_{t'})$, then $D_t = D_{t'}$ and by induction we have $L_{t'}^{live} = \{\ell \in L_{t'} : V_{t'}^\ell \cap D_{t'} = \emptyset\}$. As all other labels stay the same, the only interesting labels are i and j. Since $i \notin L_t$, there is nothing to prove for label i. We have that $V_t^j = V_{t'}^i \cup V_{t'}^j$ and, by niceness of μ, we have $i, j \in L_{t'}$ and $j \in L_{t'}^{live} \iff i, j \in L_{t'}^{live}$. Now,

$$j \in L_t^{live} \iff i, j \in L_{t'}^{live} \iff V_{t'}^i \cap D_{t'} = \emptyset \wedge V_{t'}^j \cap D_{t'} = \emptyset \iff V_t^j \cap D_t = \emptyset$$

  where the last equivalence follows from $D_t = D_{t'}$ and $V_t^j = V_{t'}^i \cup V_{t'}^j$.
- If $t = \eta_{i,j}(G_{t'})$, then $V_t^\ell = V_{t'}^\ell$ for all $\ell \in L_t = L_{t'}$ and $D_{t'} \subseteq D_t$ and hence the forward implication follows from the statement at t'. If t is not the child of a dead node, then we even have $D_t = D_{t'}$, so also the reverse implication follows from the statement at t'.
- If t is a dead node, then either the child or grandchild of t is the join node $t^* = \eta_{i,j}(G_{t'})$ that caused t to exist. By induction, $L_{t'}^{live} = \{\ell \in L_{t'} : V_{t'}^\ell \cap D_{t'} = \emptyset\}$. We have that $V_{t^*}^\ell = V_t^\ell = V_{t'}^\ell$ for every $\ell \in L_t = L_{t'}$, $D_t \setminus D_{t'} = D_{t^*} \setminus D_{t'} \subseteq V_t^i \cup V_t^j$ and $L_t^{live} = L_{t'}^{live} \setminus X$, where $\emptyset \neq X \subseteq \{i, j\}$ is the set of labels that were removed from the live labels between t^* and t. Since $L_t^{live} \setminus \{i, j\} = L_{t'}^{live} \setminus \{i, j\}$, the equivalence holds for all $\ell \in L_t \setminus \{i, j\}$ by induction. For $\ell \in X = L_{t'}^{live} \setminus L_t^{live}$, we have $V_t^\ell \subseteq D_t \setminus D_{t'}$ by construction of dead nodes and hence $V_t^\ell \cap D_t \neq \emptyset$, therefore the forward implication holds for all labels $\ell \in L_t$ at node t. Finally, if t is not the child of a dead node, then we have removed all labels $\ell \in L_t$ with $V_t^\ell \subseteq D_{t^*} \setminus D_{t'} = D_t \setminus D_{t'}$, so, in this case, the reverse implication holds for all labels $\ell \in L_t$ as well.

- If t is a union node, then $V_t^\ell = V_{t_1}^\ell \cup V_{t_2}^\ell$ for all $\ell \in [k]$ and $D_t = D_{t_1} \cup D_{t_2}$. If $\ell \in L_{t_1} \triangle L_{t_2} \subseteq L_t$, then suppose without loss of generality that $\ell \in L_{t_1} \setminus L_{t_2}$, hence $V_{t_1}^\ell = V_t^\ell$ and $V_{t_2}^\ell = \emptyset$, so we see that

$$\begin{aligned}\ell \in L_t^{live} &\iff \ell \in L_{t_1}^{live} \iff V_{t_1}^\ell \cap D_{t_1} = \emptyset \iff V_{t_1}^\ell \cap (D_{t_1} \cup D_{t_2}) = \emptyset \\ &\iff V_t^\ell \cap D_t = \emptyset,\end{aligned}$$

  where the third equivalence uses $V_{t_1}^\ell \cap D_{t_2} = \emptyset$. If on the other hand $\ell \in L_{t_1} \cap L_{t_2} \subseteq L_t$, then it follows from Lemma 5.1.6 that $V_{t_1}^\ell \cap D_{t_1} = \emptyset$ if and only if $V_{t_2}^\ell \cap D_{t_2} = \emptyset$. Therefore,

$$\ell \in L_t^{live} \iff \ell \in L_{t_1}^{live} \cup L_{t_2}^{live} \iff V_{t_1}^\ell \cap D_{t_1} = \emptyset \wedge V_{t_2}^\ell \cap D_{t_2} = \emptyset \iff V_t^\ell \cap D_t = \emptyset,$$

  where the last equivalence follows from $V_{t_1}^\ell \cap D_{t_2} = \emptyset = V_{t_2}^\ell \cap D_{t_1}$. □

Lemma 5.1.9. *Let μ be a nice k-expression of $G = (V, E)$. For every node $t \in V(\hat{T}_\mu)$, the set of dead labels L_t^{dead} satisfies the following recurrences:*

$$L_t^{dead} = \begin{cases} \emptyset & \text{if } t = \ell(v), \\ L_{t'}^{dead} \setminus \{i\} & \text{if } t = \rho_{i\to j}(G_{t'}), \\ L_{t'}^{dead} & \text{if } t = \eta_{i,j}(G_{t'}), \\ L_{t'}^{dead} \cup \{\ell\} & \text{if } t = \bot_\ell(G_{t'}), \\ L_{t_1}^{dead} \cup L_{t_2}^{dead} & \text{if } t = G_{t_1} \oplus G_{t_2}. \end{cases}$$

Proof. For every $t \in V(\hat{T}_\mu)$, the set of nonempty labels L_t is the disjoint union of L_t^{live} and L_t^{dead} by definition of L_t^{dead}. For join and relabel nodes, we have that $L_t = L_{t'}$, so the recurrences directly follow from the recurrences for L_t^{live}. For introduce nodes $t = \ell(v)$, we have $L_t = L_t^{live} = \{\ell\}$ and hence $L_t^{dead} = \emptyset$. For relabel nodes $t = \rho_{i\to j}(G_{t'})$, we have $L_t = L_{t'} \setminus \{i\}$ and hence the recurrence follows.

For a union node $t = G_{t_1} \oplus G_{t_2}$, we have by Lemma 5.1.8 that $L_{t_i}^{live} = \{\ell \in L_{t_i} : V_{t_i}^\ell \cap D_{t_i} = \emptyset\}$ for $i \in \{1, 2\}$. Hence, for any $\ell \in L_{t_1} \cap L_{t_2} \subseteq L_t$, we have $\ell \in L_{t_1}^{live} \iff \ell \in L_{t_2}^{live}$ by irredundancy of μ. Therefore, we compute

$$\begin{aligned} L_t^{dead} &= L_t \setminus L_t^{live} = (L_{t_1} \cup L_{t_2}) \setminus (L_{t_1}^{live} \cup L_{t_2}^{live}) \\ &= (L_{t_1} \setminus (L_{t_1}^{live} \cup L_{t_2}^{live})) \cup (L_{t_2} \setminus (L_{t_1}^{live} \cup L_{t_2}^{live})) \\ &= (L_{t_1}^{dead} \cap (L_{t_1} \setminus L_{t_2}^{live})) \cup (L_{t_2}^{dead} \cap (L_{t_2} \setminus L_{t_1}^{live})) \\ &= L_{t_1}^{dead} \cup L_{t_2}^{dead}, \end{aligned}$$

since the preceding argumentation shows that $L_{t_1}^{dead} \cap L_{t_2}^{live} = \emptyset = L_{t_2}^{dead} \cap L_{t_1}^{live}$. □

Since the set D_t never shrinks when going up the augmented syntax tree $\hat{T}_\mu$, a dead vertex can never turn live again. Even though the relation between D_t and L_t^{live} does not always hold in both directions, cf. Lemma 5.1.8, the next lemma still shows that a vertex can never switch from a dead label back to a live label.

Lemma 5.1.10. *Let μ be a nice k-expression of $G = (V, E)$ and $v \in V$ a vertex. For any node $t \in V(\hat{T}_\mu)$ such that $v \in V_t^\ell$ with $\ell \in L_t^{dead}$, we have for any ancestor $t^* \in V(\hat{T}_\mu)$ and $\ell^* \in L_{t^*}$ with $v \in V_{t^*}^{\ell^*}$ that $\ell^* \in L_{t^*}^{dead}$.*

Proof. Consider the recurrences of Lemma 5.1.9 and note that for join nodes and dead nodes t, we have that $L_{t'}^{dead} \subseteq L_t^{dead}$, where t' is the child of t. For union nodes t, we have that $L_{t_i}^{dead} \subseteq L_t^{dead}$, where t_i, $i \in [2]$, are the children of t. It remains to consider relabel nodes $t = \rho_{i \to j}(G_{t'})$, where t' is the child of t. Here we have that $L_t^{dead} = L_{t'}^{dead} \setminus \{i\}$, because label i is empty at t. If we have $i \in L_{t'}^{dead}$, then also $j \in L_{t'}^{dead}$ and $j \in L_t^{dead}$ by irredundancy of μ and Lemma 5.1.8. Since $V_t^j = V_{t'}^i \cup V_{t'}^j$, this shows that also at relabel nodes no vertex can switch from a dead label to a live label. □

Handling Union Nodes. At a union node of a k-expression, one often has to efficiently compute a convolution-like recurrence for the dynamic programming algorithm. The first step is to handle the labels that are nonempty at only one of the children of the union node. For these, the computation is usually trivial and the remaining part is to design a fast convolution algorithm tailored to the problem for the labels which are nonempty at both children. To encapsulate this splitting of the label set, we make the following definition.

Definition 5.1.11. Let μ be a nice k-expression of $G = (V, E)$ and $t \in V(\hat{T}_\mu)$ be a union node, i.e., $t = G_{t_1} \oplus G_{t_2}$. The *union-split at t* of a function $f \colon L_t^{live} \to \mathbf{States}$, where $\mathbf{States}$ is some finite set, are the functions $f_{t,1} = f|_{\tilde{L}_{t,1}}$, $f_{t,2} = f|_{\tilde{L}_{t,2}}$, $f_{t,12} = f|_{\tilde{L}_{t,12}}$, where $\tilde{L}_{t,1} = L_{t_1}^{live} \setminus L_{t_2}$, $\tilde{L}_{t,2} = L_{t_2}^{live} \setminus L_{t_1}$, $\tilde{L}_{t,12} = L_{t_1}^{live} \cap L_{t_2}^{live}$.

Given this definition of the union-split, it is conceivable that nonempty labels that are live at one child but dead at the other are not accounted for by any of the sets $\tilde{L}_{t,1}, \tilde{L}_{t,2}, \tilde{L}_{t,12}$. The next lemma shows that such labels cannot exist for nice clique-expressions.

Lemma 5.1.12. *Let μ be a nice k-expression of $G = (V, E)$ and $t \in V(\hat{T}_\mu)$ be a union node, i.e., $t = G_{t_1} \oplus G_{t_2}$. We have that $L_{t_1}^{live} \cap L_{t_2}^{dead} = \emptyset = L_{t_1}^{dead} \cap L_{t_2}^{live}$ and the sets $\tilde{L}_{t,1}$, $\tilde{L}_{t,2}$, and $\tilde{L}_{t,12}$ partition L_t^{live}.*

Proof. The three sets are clearly disjoint by definition. Since $L_t^{live} = L_{t_1}^{live} \cup L_{t_2}^{live}$, it suffices to argue that $\tilde{L}_{t,1} = L_{t_1}^{live} \setminus L_{t_2}^{live}$ and $\tilde{L}_{t,2} = L_{t_2}^{live} \setminus L_{t_1}^{live}$. Note that $\tilde{L}_{t,1} = L_{t_1}^{live} \setminus L_{t_2}^{live}$ is equivalent to $L_{t_1}^{live} \cap L_{t_2}^{dead} = \emptyset$ and similarly for the other equality. Without loss of generality, we consider $\tilde{L}_{t,1}$. We have that $\tilde{L}_{t,1} = L_{t_1}^{live} \setminus L_{t_2} \subseteq L_{t_1}^{live} \setminus L_{t_2}^{live}$, since $L_{t_2}^{live} \subseteq L_{t_2}$. For the other direction, note that any label $\ell \in L_{t_1}^{live} \cap L_{t_2}$ must be contained in $L_{t_2}^{live}$, as otherwise we would have $V_{t_1}^\ell \cap D_{t_1} = \emptyset$ and $V_{t_2}^\ell \subseteq D_{t_2}$ by Lemma 5.1.8, which implies $V_t^\ell \cap D_t = V_{t_2}^\ell \notin \{\emptyset, V_t^\ell\}$ contradicting Lemma 5.1.6. □

5.2 Fast Convolution Algorithms

In this section, we develop two fast convolution algorithms that we use to speed up the computations at the union nodes in the clique-width algorithms. For many clique-width algorithms the possible states of a label class are described by a set family. In such cases, we have to update the label states by a set-union-like operation when combining two partial

solutions at a union node. When the set family is simply the power set of some ground set, then standard fast subset convolution algorithms are usually sufficient. However, after optimizing the number of states, the remaining set family of label states does not have such a nice structure anymore and we cannot directly apply the standard algorithms. Two cases occur for the considered problems. The first case is that the remaining set family is a *trimmed power set*, where we remove a superset-closed family from the top of the power set and a subset-closed family from the bottom of the power set, so the remaining set family $\mathcal{F}$ has no gaps in the sense that $W, S \in \mathcal{F}$ and $W \subseteq T \subseteq S$ implies $T \in \mathcal{F}$. In the second case, the set family $\mathcal{F}$ can contain gaps, but forms a *lattice*, i.e., for every $S, T \in \mathcal{F}$ there exists a *join* $S \vee T \in \mathcal{F}$ with $S \cup T \subseteq S \vee T$ and a *meet* $S \wedge T \in \mathcal{F}$ with $S \wedge T \subseteq S \cap T$. We develop fast convolution algorithms for both cases.

5.2.1 Trimmed Subset Convolution

In this subsection, we provide a convolution algorithm to quickly compute the *cover product* which occurs at the union node for, e.g., the Connected Vertex Cover[clique-width] and Connected Odd Cycle Transversal[clique-width] algorithm. As an example, we consider the states of the Connected Vertex Cover algorithm, which are given[1] by $\mathbf{States} = \mathcal{P}(\Omega) \setminus \{\emptyset, \Omega\}$, where $\Omega = \{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$. Essentially, we are given two tables $A, B \colon \mathbf{States}^{[k]} \to \mathbb{Z}_2$ and want to compute the following *cover product* $A \otimes_c B \colon \mathbf{States}^{[k]} \to \mathbb{Z}_2$ in time $\mathcal{O}^*(6^k) = \mathcal{O}^*(|\mathbf{States}|^k)$:

$$(A \otimes_c B)(f) = \sum_{\substack{f_1, f_2 \in \mathbf{States}^{[k]} \colon \\ f_1 \cup f_2 = f}} A(f_1) B(f_2),$$

where the union is *componentwise*, i.e., $(f_1 \cup f_2)(\ell) = f_1(\ell) \cup f_2(\ell)$ for all $\ell \in [k]$. If not for the exclusion of $\emptyset$ and Ω, a standard application of the fast Zeta and Möbius transform [47] would be sufficient. However, this results only in a running time of $\mathcal{O}^*(8^k)$. To obtain the improved running time, we trim the computations of the fast Zeta and Möbius transform from above and below. By ordering the subsets of Ω by their size and performing the computation along this ordering, we can start only with the relevant subsets, i.e., exclude $\emptyset$, and can stop the computation before we reach Ω itself. All these ideas were used by Björklund et al. [15], but their presentation is not suited to our setting, instead, we provide a suitable self-contained presentation.

We first switch to a more standard notation and consider tables defined over some set family $\mathcal{F} \subseteq \mathcal{P}(U)$ and finite universe U instead of over functions from $[k]$ to $\mathbf{States}$. A given function $f \colon [k] \to \mathbf{States}$ is transformed to the set $S_f = \{(i, \mathbf{s}) : i \in [k], \mathbf{s} \in f(i)\} = \bigcup_{i \in [k]} \{i\} \times f(i) \subseteq \bigcup_{i \in [k]} \{i\} \times \Omega =: U$ and the resulting set family is given by $\mathcal{F} = \{S \subseteq U : 1 \leq |S \cap (\{i\} \times \Omega)| \leq 2 \ \forall i \in [k]\}$. We will see that this set family can be written in a special form that allows us to provide trimmed algorithms for the Zeta and Möbius transform.

For now, we return to the general setting. Let U be some fixed universe and let $\mathcal{F} \subseteq \mathcal{P}(U)$ be some set family over U. The *upward closure* $\uparrow(\mathcal{F}) \subseteq \mathcal{P}(U)$ of $\mathcal{F}$ is given by $\uparrow(\mathcal{F}) = \{S \subseteq$

[1] Their precise meaning is irrelevant here and will be discussed in Section 5.3.

U : there is a $T \in \mathcal{F}$ s.t. $T \subseteq S\}$; a set family $\mathcal{F}$ is closed under supersets if and only if $\mathcal{F} = \uparrow(\mathcal{F})$. We say that a set family $\mathcal{F}$ is a *closure difference* if there are set families $\mathcal{F}_+$ and $\mathcal{F}_-$ such that $\mathcal{F} = \uparrow(\mathcal{F}_+) \setminus \uparrow(\mathcal{F}_-)$. Observe that $\uparrow(\emptyset) = \emptyset$ and hence $\uparrow(\mathcal{F}_+)$ is also a closure difference. Closure differences are precisely the set families that contain no holes in the sense of the following lemma, which will be important to prove the properties of the various transforms.

Lemma 5.2.1. *A set family $\mathcal{F} \subseteq \mathcal{P}(U)$ is a closure difference if and only if it satisfies the following* interval property*: for all sets $W, T, S \subseteq U$ with $W \subseteq T \subseteq S$ and $W, S \in \mathcal{F}$, we must also have $T \in \mathcal{F}$.*

Proof. We first show that the interval property holds when $\mathcal{F}$ is a closure difference. So, let $\mathcal{F} = \uparrow(\mathcal{F}_+) \setminus \uparrow(\mathcal{F}_-)$ and $W \subseteq T \subseteq S$ with $W \in \mathcal{F}$ and $S \in \mathcal{F}$. There exists a set $X \in \mathcal{F}_+$ with $X \subseteq W$, since $W \in \uparrow(\mathcal{F}_+)$, and for all sets $Y \in \mathcal{F}_-$ we have $Y \not\subseteq S$, since $S \notin \uparrow(\mathcal{F}_-)$. Hence, we have $T \in \uparrow(\mathcal{F}_+)$ due to $X \subseteq W \subseteq T$ and no $Y \in \mathcal{F}_-$ can satisfy $Y \subseteq T$, else we would also have $Y \subseteq S$. Therefore, $T \in \uparrow(\mathcal{F}_+) \setminus \uparrow(\mathcal{F}_-) = \mathcal{F}$.

For the other direction, we first show that $\uparrow(\mathcal{F}) \setminus \mathcal{F}$ is an upward closure, i.e., closed under taking supersets. Suppose, for sake of contradiction, that $T \in \uparrow(\mathcal{F}) \setminus \mathcal{F}$ and $T \subseteq S$, but $S \notin \uparrow(\mathcal{F}) \setminus \mathcal{F}$. Since $T \in \uparrow(\mathcal{F})$, there exists some $W \in \mathcal{F}$ with $W \subseteq T \subseteq S$, hence also $S \in \uparrow(\mathcal{F})$. From $S \notin \uparrow(\mathcal{F}) \setminus \mathcal{F}$, it then follows that $S \in \mathcal{F}$. Finally, since $\mathcal{F}$ satisfies the interval property, we must then have $T \in \mathcal{F}$, thus contradicting that $T \in \uparrow(\mathcal{F}) \setminus \mathcal{F}$. So, $\uparrow(\mathcal{F}) \setminus \mathcal{F} = \uparrow(\mathcal{F}_-)$ for some family $\mathcal{F}_-$ and we obtain $\mathcal{F} = \uparrow(\mathcal{F}) \setminus (\uparrow(\mathcal{F}) \setminus \mathcal{F}) = \uparrow(\mathcal{F}) \setminus \uparrow(\mathcal{F}_-)$. □

Given a table $A \colon \mathcal{F} \to R$, where R is some commutative ring with unit, the *Zeta transform* $\zeta A \colon \mathcal{F} \to R$, the *Möbius transform* $\mu A \colon \mathcal{F} \to R$, and the *odd-negation transform* $\sigma A \colon \mathcal{F} \to R$ are given by

$$(\zeta A)(S) = \sum_{\substack{T \in \mathcal{F} \colon \\ T \subseteq S}} A(T), \qquad (\mu A)(S) = \sum_{\substack{T \in \mathcal{F} \colon \\ T \subseteq S}} (-1)^{|S \setminus T|} A(T), \qquad (\sigma A)(S) = (-1)^{|S|} A(S).$$

All of these transforms can be viewed as operators on the space of functions from $\mathcal{F}$ to R. Given two tables $A, B \colon \mathcal{F} \to R$, their *cover product* $A \otimes_c B \colon \mathcal{F} \to R$ is given by

$$(A \otimes_c B)(S) = \sum_{\substack{T_1, T_2 \in \mathcal{F} \colon \\ T_1 \cup T_2 = S}} A(T_1) B(T_2).$$

Moreover, we let $A \cdot B \colon \mathcal{F} \to R$ denote the pointwise multiplication of the two tables A and B, i.e., $(A \cdot B)(S) = A(S)B(S)$. We will now prove several properties of these transforms that allow us to design a fast algorithm for computing the cover product.

Lemma 5.2.2. *Let $\mathcal{F}$ be a closure difference and R a commutative ring with unit. The following statements are true:*

1. $\sigma\zeta\sigma = \mu$, $\sigma\mu\sigma = \zeta$,
2. $\mu\zeta = \zeta\mu = \mathrm{id}$, *where* id *is the identity transform,*
3. $\zeta(A \otimes_c B) = (\zeta A) \cdot (\zeta B)$, *for any two tables $A, B \colon \mathcal{F} \to R$.*

Proof. We adapt the proofs of Cygan et al. [47] to our setting and make note of the proof steps where we use that $\mathcal{F}$ is a closure difference.

1. We compute for every table $A\colon \mathcal{F} \to R$ and set $S \in \mathcal{F}$:

$$\begin{aligned}(\sigma\zeta\sigma A)(S) &= (-1)^{|S|}\sum_{\substack{T\in\mathcal{F}:\\ T\subseteq S}}(-1)^{|T|}A(T) = \sum_{\substack{T\in\mathcal{F}:\\ T\subseteq S}}(-1)^{|S|+|T|}A(T) = \sum_{\substack{T\in\mathcal{F}:\\ T\subseteq S}}(-1)^{|S\setminus T|}A(T)\\ &= (\mu A)(S),\end{aligned}$$

   where the third equality follows from $|S| + |T| = 2|S \cap T| + |S \setminus T|$. Since $\sigma\sigma = \mathrm{id}$, it also follows that $\sigma\mu\sigma = \sigma\sigma\zeta\sigma\sigma = \zeta$.
2. From the previous statement it follows that $\mu\zeta = \sigma\zeta\sigma\zeta$. Again, we compute for every table $A\colon \mathcal{F} \to R$ and set $S \in \mathcal{F}$:

$$\begin{aligned}(\sigma\zeta\sigma\zeta A)(S) &= (-1)^{|S|}\sum_{\substack{T\in\mathcal{F}:\\ T\subseteq S}}(\sigma\zeta A)(T) = (-1)^{|S|}\sum_{\substack{T\in\mathcal{F}:\\ T\subseteq S}}(-1)^{|T|}\sum_{\substack{W\in\mathcal{F}:\\ W\subseteq T}}A(W)\\ &= (-1)^{|S|}\sum_{\substack{W\in\mathcal{F}:\\ W\subseteq S}}A(W)\sum_{\substack{T\in\mathcal{F}:\\ W\subseteq T\subseteq S}}(-1)^{|T|} = A(S) + (-1)^{|S|}\sum_{\substack{W\in\mathcal{F}:\\ W\subsetneq S}}A(W)\sum_{\substack{T\in\mathcal{F}:\\ W\subseteq T\subseteq S}}(-1)^{|T|}\\ &= A(S),\end{aligned}$$

   where the last equality follows from the fact that $\sum_{T\in\mathcal{F}\colon W\subseteq T\subseteq S}(-1)^{|T|} = 0$ whenever $W \subsetneq S$, since we can pick some $x \in S \setminus W$ and pair every T with $T \triangle \{x\}$ which yields a fixpoint-free sign-reversing involution. Note that this step relies on Lemma 5.2.1 (applied to $W \subseteq T \triangle \{x\} \subseteq S$). For an arbitrary set family $\mathcal{F}'$, it does not necessarily hold that $T \triangle \{x\} \in \mathcal{F}'$.

   The remaining equality follows from $\zeta\mu = \zeta\sigma\zeta\sigma = \sigma\mu\sigma\sigma\zeta\sigma = \sigma\mu\zeta\sigma = \sigma\sigma = \mathrm{id}$.
3. We compute for every two tables $A, B\colon \mathcal{F} \to R$ and set $S \in \mathcal{F}$:

$$\begin{aligned}(\zeta(A \otimes_c B))(S) &= \sum_{\substack{W\in\mathcal{F}:\\ W\subseteq S}}\sum_{\substack{T_1,T_2\in\mathcal{F}:\\ T_1\cup T_2=W}}A(T_1)B(T_2) = \sum_{\substack{T_1,T_2\in\mathcal{F}:\\ T_1\cup T_2\subseteq S}}A(T_1)B(T_2)\\ &= \sum_{\substack{T_1\in\mathcal{F}:\\ T_1\subseteq S}}\sum_{\substack{T_2\in\mathcal{F}:\\ T_2\subseteq S}}A(T_1)B(T_2) = \left(\sum_{\substack{T\in\mathcal{F}:\\ T\subseteq S}}A(T)\right)\left(\sum_{\substack{T\in\mathcal{F}:\\ T\subseteq S}}B(T)\right)\\ &= ((\zeta A)\cdot(\zeta B))(S),\end{aligned}$$

   where the second equality again relies on Lemma 5.2.1 (applied to $T_1 \subseteq T_1 \cup T_2 \subseteq S$). For an arbitrary set family $\mathcal{F}'$, we might have $T_1 \cup T_2 \notin \mathcal{F}'$ which would therefore not be summed over on the left-hand side. □

To proceed with the algorithmic part, we require that $\mathcal{F}$ satisfies some algorithmic requirements. We say that $\mathcal{F} \subseteq \mathcal{P}(U)$ is *efficiently listable* if there is an algorithm that outputs all members of $\mathcal{F}$ in time $\mathcal{O}^*(|\mathcal{F}|) = |\mathcal{F}||U|^{\mathcal{O}(1)}$. Each member of $\mathcal{F}$ is represented as a bitstring and we can assume that the output of the listing algorithm is in sorted order. Hence, after listing $\mathcal{F}$, we can decide for a set $S \subseteq U$ whether $S \in \mathcal{F}$ or not by binary search in time $\mathcal{O}(\log|\mathcal{F}|) = \mathcal{O}(|U|)$. A table $A\colon \mathcal{F} \to R$ is given by listing the values $A(S)$, $S \in \mathcal{F}$, in the same order as the listing of $\mathcal{F}$.

In the following time analysis, we separate the impact of the ring operations from the rest of the algorithms, i.e., we only count the number of performed ring operations and bound the time spent on the remainder of the algorithm.

Corollary 5.2.3. *Let $\mathcal{F}$ be an efficiently listable closure difference and R be a commutative ring with unit. If the Zeta transform ζA and the Möbius transform μA of a table $A\colon \mathcal{F} \to R$ can be computed in $\mathcal{O}(|\mathcal{F}||U|)$ ring operations and additional time $\mathcal{O}^*(|\mathcal{F}|)$, then the cover product $A \otimes_c B$ can be computed in $\mathcal{O}(|\mathcal{F}||U|)$ ring operations and additional time $\mathcal{O}^*(|\mathcal{F}|)$.*

Proof. We make use of Lemma 5.2.2. Given tables A and B, we first compute ζA and ζB and then the pointwise multiplication $(\zeta A) \cdot (\zeta B)$. By definition, the pointwise multiplication can be computed in $\mathcal{O}(|\mathcal{F}|)$ ring operations and additional time $\mathcal{O}^*(|\mathcal{F}|)$ for listing all members of $\mathcal{F}$. Finally, we compute the Möbius transform of the pointwise multiplication. By Lemma 5.2.2, we have $\mu((\zeta A) \cdot (\zeta B)) = \mu\zeta(A \otimes_c B) = A \otimes_c B$. By assumption, every step takes $\mathcal{O}(|\mathcal{F}||U|)$ ring operations and additional time $\mathcal{O}^*(|\mathcal{F}|)$, hence the statement follows. □

Theorem 5.2.4. *Let $\mathcal{F} = \uparrow(\mathcal{F}_+) \setminus \uparrow(\mathcal{F}_-)$ be an efficiently listable closure difference and R be a commutative ring with unit. Given a table $A\colon \mathcal{F} \to R$, the Zeta transform ζA and the Möbius transform μA can be computed in $\mathcal{O}(|\mathcal{F}||U|)$ ring operations and additional time $\mathcal{O}^*(|\mathcal{F}|)$.*

Proof. We modify the algorithm of Björklund et al. [15, Algorithm Z] for the Zeta transform to make it suitable for our setting. Let $n = |U|$ be the size of the universe and without loss of generality assume $U = [n]$. The algorithm maintains $n+1$ set families $0, \ldots, \mathcal{L}_n \subseteq \mathcal{F}$, where each $\mathcal{L}_i$ only contains sets of size i. For all $j \in [0, n]$ and $S \in \mathcal{F}$, we will compute auxiliary values

$$A_j(S) = \sum_{\substack{T \in \mathcal{F}\colon T \cap [j] \subseteq S \cap [j],\\ T \cap [j+1,n] = S \cap [j+1,n]}} A(T).$$

Clearly, $A_0(S) = A(S)$ and $A_n(S) = \zeta A(S)$. Also, one can verify that the recurrence $A_j(S) = [(j \in S) \wedge (S \setminus \{j\} \in \mathcal{F})]A_{j-1}(S \setminus \{j\}) + A_{j-1}(S)$ holds for all $j \in [n]$ and $S \in \mathcal{F}$. We highlight the case that $j \in S$ but $S \setminus \{j\} \notin \mathcal{F}$; here every $T \in \mathcal{F}$ with $T \subseteq S$ must satisfy $j \in T$, otherwise we would obtain $S \setminus \{j\} \in \mathcal{F}$ by Lemma 5.2.1, hence $A_j(S) = A_{j-1}(S)$.

Algorithm 1: Fast Zeta transform

```
for S ∈ F do insert S into L_{|S|};
for r = 0, 1, ..., n do
    while L_r is not empty do
        Select any S ∈ L_r and remove it from L_r;
        A_0(S) := A(S);
        for j = 1, ..., n do
            A_j(S) := [(j ∈ S) ∧ (S \ {j} ∈ F)]A_{j-1}(S \ {j}) + A_{j-1}(S);
return A_n;
```

Clearly, Algorithm 1 considers every set $S \in \mathcal{F}$ exactly once and due to the ordering by cardinality only accesses already computed values. Algorithm 1 correctly computes the Zeta

transform $\zeta A = A_n$ by the previous considerations. Ring operations are only performed in line 7, namely at most two per execution of line 7. Line 7 is executed exactly n times for every element of $\mathcal{F}$, hence Algorithm 1 performs in total $\mathcal{O}(|\mathcal{F}|n)$ ring operations. The additional time is dominated by the listing of $\mathcal{F}$ in line 1 and the $\mathcal{O}(|\mathcal{F}|n)$ membership queries incurred by line 7, thus leading to additional time $\mathcal{O}^*(|\mathcal{F}|)$.

To compute the Möbius transform μA, we use the equation $\mu = \sigma\zeta\sigma$ from Lemma 5.2.2. Hence, we first compute σA, then apply Algorithm 1 to σA to obtain $\zeta\sigma A$, and finally we apply σ again. Since $(-1)^k = 1$ if k is even and $(-1)^k = -1$ if k is odd, we can apply σ in $\mathcal{O}(|\mathcal{F}|)$ ring operations and additional time $\mathcal{O}^*(|\mathcal{F}|)$ due to listing, therefore the statement follows. □

Componentwise Union to Set Union. Fix a natural number k and universe U. We define the *projection* $\pi_U \colon [k] \times U \to U$, $(i, u) \mapsto u$, and for any $i \in [k]$ and set $S \subseteq U$, we define $\pi_U^i(S) := \pi_U(S \cap (\{i\} \times U))$. Given a set family $\mathcal{F} \subseteq \mathcal{P}(U)$, we construct the set family $k\mathcal{F} := \{S \subseteq [k] \times U : \pi_U^i(S) \in \mathcal{F} \text{ for all } i \in [k]\}$. Given a function $f \colon [k] \to \mathcal{F}$, we construct the set $k\mathcal{F} \ni S_f := \bigcup_{i\in[k]} \{i\} \times f(i) = \{(i, x) \in [k] \times U : x \in f(i)\}$. This is a bijection and the construction turns componentwise union of functions $[k] \to \mathcal{F}$ into union of sets in $k\mathcal{F}$, i.e., for functions $f, f_1, f_2 \colon [k] \to \mathcal{F}$, we have $f(i) = f_1(i) \cup f_2(i)$ for all $i \in [k]$ if and only if $S_f = S_{f_1} \cup S_{f_2}$. Hence, given two tables $A, B \colon \mathcal{F}^{[k]} \to R$, the *componentwise cover product* $A \otimes_{comp} B \colon \mathcal{F}^{[k]} \to R$ can be reduced to a standard cover product, i.e.,

$$(A \otimes_{comp} B)(f) := \sum_{\substack{f_1, f_2 \in \mathcal{F}^{[k]} : \\ f_1(i) \cup f_2(i) = f(i)\ \forall i \in [k]}} A(f_1)B(f_2) = \sum_{\substack{S_{f_1}, S_{f_2} \in k\mathcal{F} : \\ S_{f_1} \cup S_{f_2} = S_f}} A(f_1)B(f_2) = (A' \otimes_c B')(S_f),$$

where $A', B' \colon k\mathcal{F} \to R$ with $A'(S_f) := A(f)$ and $B'(S_f) := B(f)$, which is well-defined since $f \mapsto S_f$ is a bijection. Furthermore, the following lemma shows that we can apply the fast convolution algorithms to $k\mathcal{F}$ if $\mathcal{F}$ is a closure difference.

Lemma 5.2.5. *Let $\mathcal{F} \subseteq \mathcal{P}(U)$ be a set family and k be some natural number. If $\mathcal{F}$ is a closure difference, then also the set family $k\mathcal{F} \subseteq \mathcal{P}([k] \times U)$ is a closure difference.*

Proof. We first argue that $k{\uparrow}(\mathcal{F}) = {\uparrow}(k\mathcal{F})$ for all set families $\mathcal{F}$ and natural numbers k. We have $S \in k{\uparrow}(\mathcal{F})$ if and only if there exist $T_i \in \mathcal{F}$ such that $\{i\} \times T_i \subseteq S \cap (\{i\} \times U)$ for all $i \in [k]$. Setting $T = \bigcup_{i\in[k]} \{i\} \times T_i \in k\mathcal{F}$, we see that this is equivalent to the existence of some $T \subseteq S$ which holds if and only if $S \in {\uparrow}(k\mathcal{F})$.

Let $\mathcal{F} = {\uparrow}(\mathcal{F}_+) \setminus {\uparrow}(\mathcal{F}_-)$ be a closure difference. We compute

$$\begin{aligned}
k\mathcal{F} &= \{S \subseteq [k] \times U : \pi_U^i(S) \in \mathcal{F} \text{ for all } i \in [k]\} \\
&= \{S \subseteq [k] \times U : \pi_U^i(S) \in ({\uparrow}(\mathcal{F}_+) \setminus {\uparrow}(\mathcal{F}_-)) \text{ for all } i \in [k]\} \\
&= k{\uparrow}(\mathcal{F}_+) \setminus \{S \subseteq [k] \times U : \text{ there is an } i \in [k] \text{ s.t. } \pi_U^i(S) \in {\uparrow}(\mathcal{F}_-)\} \\
&= {\uparrow}(k\mathcal{F}_+) \setminus \{S \subseteq [k] \times U : \text{ there is an } i \in [k] \text{ s.t. } \pi_U^i(S) \in {\uparrow}(\mathcal{F}_-)\}
\end{aligned}$$

and notice that the second set in the last line is clearly closed under taking supersets and hence $k\mathcal{F}$ is a closure difference, too. □

Theorem 5.2.6. *Let $\mathcal{F} \subseteq \mathcal{P}(U)$ be a fixed efficiently listable closure difference and R be a commutative ring with unit. Given tables $A, B\colon \mathcal{F}^{[k]} \to R$, their componentwise cover product $A \otimes_{comp} B$ can be computed in $\mathcal{O}(k|\mathcal{F}|^k|U|)$ ring operations and additional time $\mathcal{O}^*(|\mathcal{F}|^k)$.*

Proof. We construct the set family $k\mathcal{F} \subseteq \mathcal{P}([k] \times U)$ which is a closure difference by Lemma 5.2.5. Since $\mathcal{F}$ is efficiently listable, also $k\mathcal{F}$ can be efficiently listed in lexicographic order. We invoke Theorem 5.2.4 and Corollary 5.2.3 with $k\mathcal{F}$ to compute the cover product over $k\mathcal{F}$ in $\mathcal{O}(k|\mathcal{F}|^k|U|)$ ring operations and additional time $\mathcal{O}^*(|\mathcal{F}|^k)$. By the preceding discussion, the cover product over $k\mathcal{F}$ yields the componentwise cover product $A \otimes_{comp} B$ over $\mathcal{F}$. □

5.2.2 Lattice-based Convolution

A *poset* is a pair $(\mathcal{L}, \preceq)$ consisting of a set $\mathcal{L}$ and a binary relation $\preceq$ on $\mathcal{L}$ that is reflexive, transitive, and anti-symmetric. A *lattice* is a poset $(\mathcal{L}, \preceq)$ such that every pair $a, b \in \mathcal{L}$ has a greatest lower bound (*meet*) $a \wedge b \in \mathcal{L}$ and a least upper bound (*join*) $a \vee b \in \mathcal{L}$. Any finite lattice $(\mathcal{L}, \preceq)$ contains a $\preceq$-minimum element $\hat{0} \in \mathcal{L}$, which is obtained by taking the meet of all elements in $\mathcal{L}$ and satisfies $\hat{0} \vee a = a \vee \hat{0} = a$ for all $a \in \mathcal{L}$.

In the algorithm for Connected Dominating Set[clique-width], the family of possible states forms a lattice and at union-nodes in the clique-expression, we must compute a convolution-like product. To obtain an efficient algorithm, we will observe some lattice-theoretic properties.

The product that we are interested in can be formulated in the lattice setting as follows. Given a lattice $(\mathcal{L}, \preceq)$ and tables $A, B\colon \mathcal{L} \to \mathbb{F}$, where $\mathbb{F}$ is some field, the *$\vee$-product* $A \otimes_{\mathcal{L}} B\colon \mathcal{L} \to \mathbb{F}$ is given by $(A \otimes_{\mathcal{L}} B)(x) = \sum_{y,z \in \mathcal{L}\colon x = y \vee z} A(y)B(z)$ for every $x \in \mathcal{L}$.

Björklund et al. [16] develop an efficient algorithm for the $\vee$-product for specific lattices by designing small arithmetic circuits for the Zeta and Möbius transform, whose precise definitions we do not need here. The relevant concept is as follows; we say that an element $x \in \mathcal{L}$ of a lattice $(\mathcal{L}, \preceq)$ is *join-irreducible* if $x = a \vee b$ implies $x = a$ or $x = b$ for all $a, b \in \mathcal{L}$, otherwise x is called *join-reducible*. We denote the set of join-irreducible elements in $\mathcal{L}$ by $\mathcal{L}_\vee$. Observe that $\hat{0}$ is always join-irreducible, as otherwise $\hat{0}$ would not be the $\preceq$-minimum. We even have the stronger property that $\hat{0} = a \vee b$ implies $a = b = \hat{0}$.

We assume that a finite lattice $(\mathcal{L}, \preceq)$ is algorithmically given to us in the *join representation* [16]; we are given the set $\mathcal{L}$, where the elements of $\mathcal{L}$ represented as $\mathcal{O}(\log |\mathcal{L}|)$-bit strings, the set of join-irreducible elements $\mathcal{L}_\vee \subseteq \mathcal{L}$, and an algorithm $\mathbb{A}_{\mathcal{L}}$ that computes the join $a \vee x$ given an element $a \in \mathcal{L}$ and a join-irreducible element $x \in \mathcal{L}_\vee$.

Theorem 5.2.7 ([16])**.** *Let $(\mathcal{L}, \preceq)$ be a finite lattice given in join-representation and $A, B\colon \mathcal{L} \to \mathbb{F}$ be two tables, where $\mathbb{F}$ is some field. The $\vee$-product $A \otimes_{\mathcal{L}} B$ can be computed in $\mathcal{O}(|\mathcal{L}||\mathcal{L}_\vee|)$ field operations and calls to algorithm $\mathbb{A}_{\mathcal{L}}$ and further time $\mathcal{O}(|\mathcal{L}||\mathcal{L}_\vee|^2)$.*

Next, we analyze the lattice that occurs in the algorithm for Connected Dominating Set and derive a bound on the number of join-irreducible elements in this lattice, so that we can apply Theorem 5.2.7. The relevant lattice can be written as a power of a smaller lattice and we give a general bound for such lattices. We proceed with the relevant definitions.

Given finitely many lattices $(\mathcal{L}_i, \preceq_i)$, $i \in I$, their *direct product* $(\prod_{i\in I} \mathcal{L}_i, \preceq)$, with $(a_i)_{i\in I} \preceq (b_i)_{i\in I}$ if and only if $a_i \preceq_i b_i$ for all $i \in I$, is again a lattice; the join- and meet-operations in the direct product lattice are given by componentwise application of the corresponding operation in the constituent lattices. Given a lattice $(\mathcal{L}, \preceq)$ and $k \in \mathbb{N}$, the *kth-power* $(\mathcal{L}^k, \preceq^k)$ of $\mathcal{L}$ is the direct product of k copies of $\mathcal{L}$.

Lemma 5.2.8. *Let $(\mathcal{L}, \preceq)$ be a finite lattice and $k \in \mathbb{N}$. In the k-th power $(\mathcal{L}^k, \preceq^k)$ of $\mathcal{L}$, an element $x \in \mathcal{L}^k$ is join-irreducible if and only if $x = (\hat{0}, \dots, \hat{0})$ or there is exactly one component that is not $\hat{0}$ and this component is join-irreducible in $\mathcal{L}$. In particular, there are exactly $1 + k(|\mathcal{L}_\vee| - 1) \leq k|\mathcal{L}|$ join-irreducible elements in $\mathcal{L}^k$.*

Proof. First, we prove that every join-irreducible element x of $\mathcal{L}^k$ must have the stated form. Suppose to the contrary that $x = (x_1, \dots, x_k)$ has at least two components that are not $\hat{0}$; without loss of generality we can assume that $x_1 \neq \hat{0}$ and $x_2 \neq \hat{0}$. We compute that $x = (x_1, \dots, x_k) = (\hat{0}, x_2, x_3, \dots, x_k) \vee (x_1, \hat{0}, x_3, \dots, x_k)$ and note that $x \neq (\hat{0}, x_2, x_3, \dots, x_k)$ and $x \neq (x_1, \hat{0}, x_3, \dots, x_k)$ by assumption. Therefore, x is join-reducible in this case. Furthermore, if x contains a join-reducible component, say $x_1 = a_1 \vee b_1$ with $a_1 \neq x_1 \neq b_1$, then x is join-reducible, since $x = (a_1, x_2, \dots, x_k) \vee (b_1, x_2, \dots, x_k)$.

For the other direction, we have $x = (\hat{0}, \dots, \hat{0})$ which is the $\preceq^k$-minimum element in $\mathcal{L}^k$ and hence join-irreducible. Now, let $x = (x_1, \dots, x_k) \in \mathcal{L}^k$ have exactly one component that is not $\hat{0}$ and let this component be join-irreducible, say $x_1 \in \mathcal{L}_\vee \setminus \{\hat{0}\}$ and $x_i = \hat{0}$ for all $i \in [2, k]$. Suppose that $x = y \vee z$ with $y, z \in \mathcal{L}^k$, then we have $\hat{0} = x_i = y_i \vee z_i$ for all $i \in [2, k]$ which implies $x_i = y_i = z_i = \hat{0}$ for all $i \in [2, k]$. For $i = 1$, we see that $x_1 = y_1 \vee z_1$ implies $x_1 = y_1$ or $x_1 = z_1$ by irreducibility of x_1, and hence $x = y$ or $x = z$. □

Corollary 5.2.9. *Let $(\mathcal{L}, \preceq)$ be a finite lattice given in join-representation and k be a natural number. There is an algorithm $\mathbb{A}_{\mathcal{L}^k}$ that computes the join $a \vee x$, where $a \in \mathcal{L}^k$ and $x \in (\mathcal{L}^k)_\vee$, using one call to $\mathbb{A}_\mathcal{L}$ and further time $\mathcal{O}(k \log |\mathcal{L}|)$.*

Proof. Every element of $\mathcal{L}$ is represented by a $\mathcal{O}(\log |\mathcal{L}|)$-bit string, hence every element of $\mathcal{L}^k$ can be represented by a $\mathcal{O}(k \log |\mathcal{L}|)$-bit string. By Lemma 5.2.8, we know that the join-irreducible element $x \in \mathcal{L}^k$ has at most one component that is not $\hat{0}$. We search for this component and the corresponding component in a in time $\mathcal{O}(k \log |\mathcal{L}|)$, afterwards we call $\mathbb{A}_\vee$ to obtain the result for this component and all other components of a remain unchanged due to $a_i \vee \hat{0} = a_i$ for all $i \in [k]$. □

Corollary 5.2.10. *Let $(\mathcal{L}, \preceq)$ be a finite lattice given in join-representation and k be a natural number. Given two tables $A, B \colon \mathcal{L}^k \to \mathbb{Z}_2$, the $\vee$-product $A \otimes_{\mathcal{L}^k} B$ in $\mathcal{L}^k$ can be computed in time $\mathcal{O}(k^2 |\mathcal{L}|^{k+2})$ and $\mathcal{O}(k|\mathcal{L}|^{k+1})$ calls to algorithm $\mathbb{A}_\mathcal{L}$.*

Proof. We pipeline Theorem 5.2.7 with Lemma 5.2.8 and Corollary 5.2.9. First, observe that the field operations in $\mathbb{Z}_2$ can be performed in constant time. Secondly, every call to $\mathbb{A}_{\mathcal{L}^k}$ can be simulated by one call to $\mathbb{A}_\mathcal{L}$ and further time $\mathcal{O}(k \log |\mathcal{L}|)$ and hence these calls lead to in total $\mathcal{O}(|\mathcal{L}^k||(\mathcal{L}^k)_\vee|(k \log |\mathcal{L}|)) = \mathcal{O}(k^2 |\mathcal{L}|^{k+1} \log |\mathcal{L}|)$ further running time, which will be dominated by the rest of the algorithm. Finally, the algorithm of Theorem 5.2.7 needs further time $\mathcal{O}(|\mathcal{L}^k||(\mathcal{L}^k)_\vee|^2) = \mathcal{O}(|\mathcal{L}|^k (k|\mathcal{L}|)^2) = \mathcal{O}(k^2 |\mathcal{L}|^{k+2})$ which dominates all other computations. □

5.3 Connected Vertex Cover Algorithm

When solving Connected Vertex Cover, we only consider Connected Vertex Cover instances where the costs are polynomially bounded in the input size. Furthermore, we assume without loss of generality that G is connected and contains at least two vertices.

Given a k-expression μ for $G = (V, E)$, we can assume, after polynomial-time preprocessing, that μ is a nice k-expression by Lemma 5.1.4. We want to apply the cut-and-count-technique to solve Connected Vertex Cover in time $\mathcal{O}^*(6^k)$. Here, we use the technique of *fixing a vertex*[2], where we first pick an edge in G, branch on one of its endpoints v_*, and in this branch only consider solutions containing v_*. Furthermore, we sample a weight function $\mathbf{w}\colon V \to [2|V|]$ for the isolation lemma, cf. Lemma 3.1.4. We perform bottom-up dynamic programming along the augmented syntax tree $\hat{T}_\mu$. At every node $t \in V(\hat{T}_\mu)$, we consider the following family of partial solutions

$$\mathcal{A}_t = \{(X, (X_L, X_R)) \in \mathcal{C}(G_t) : G_t - X \text{ contains no edges and } v_* \in V_t \to v_* \in X_L\}.$$

In other words, $\mathcal{A}_t$ contains all consistently cut vertex covers of G_t such that v_* is on the left side of the cut if possible. For every $t \in V(\hat{T}_\mu)$, $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$, we define $\mathcal{A}_t^{\overline{c},\overline{w}} = \{(X, (X_L, X_R)) \in \mathcal{A}_t : \mathbf{c}(X) = \overline{c}, \mathbf{w}(X) = \overline{w}\}$. Let $\hat{r}$ denote the root node of the augmented syntax tree $\hat{T}_\mu$. By Theorem 3.1.6, it follows that there exists a connected vertex cover X of G with $\mathbf{c}(X) \leq \overline{b}$ if there exist $\overline{c} \in [0, \overline{b}]$ and $\overline{w} \in [0, \mathbf{w}(V)]$ such that $\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}$ has odd cardinality.

To facilitate the dynamic programming algorithm, we need to analyze the behavior of a partial solution $(X, (X_L, X_R)) \in \mathcal{A}_t$ with respect to a label V_t^ℓ, $\ell \in L_t$. A single vertex $v \in V_t^\ell$ can take one of the states $\Omega = \{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$, meaning respectively $v \notin X$, or $v \in X_L$, or $v \in X_R$. To check the feasibility of $(X, (X_L, X_R))$, it is sufficient to store for each label which vertex states appear and which do not, as the constraints implied by $\mathcal{A}_t$ are "CSP-like" and they can be evaluated for every join by considering all pairs of involved vertex states. This idea yields the power set $\mathcal{P}(\Omega)$ of Ω as the set of possible states for each label.

The power set $\mathcal{P}(\Omega)$ a priori yields eight different states per label. However, we can exclude the state $\emptyset$ and the state $\Omega = \{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$ from consideration. The former can be excluded since we only need to store the state for nonempty labels. The exclusion of the state $\Omega = \{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$ is more subtle: any additional incident join would lead to an infeasible solution for this state, hence only dead labels, cf. Definition 5.1.7, may take this state. We return to this issue in a moment. Since it suffices to store the states of live labels, we set $\mathbf{States} = \mathcal{P}(\Omega) \setminus \{\emptyset, \Omega\} = \{\{\mathbf{0}\}, \{\mathbf{1}_L\}, \{\mathbf{1}_R\}, \{\mathbf{0}, \mathbf{1}_L\}, \{\mathbf{0}, \mathbf{1}_R\}, \{\mathbf{1}_L, \mathbf{1}_R\}\}$.

[2] Although it can be preferable to avoid fixing a vertex as discussed in Section 3.1, the algorithm for Connected Dominating Set[clique-width] is simplified by fixing a vertex. Hence, we decide to use technique for all clique-width algorithms for sake of consistency.

Given a node $t \in V(\hat{T}_\mu)$, a *t-signature* is a function $f\colon L_t^{live} \to \mathbf{States}$. For every node $t \in V(\hat{T}_\mu)$, $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$, and t-signature f, we define

$$\begin{aligned}\mathcal{A}_t^{\overline{c},\overline{w}}(f) = \{(X,(X_L,X_R)) \in \mathcal{A}_t^{\overline{c},\overline{w}} : \mathbf{0} \in f(\ell) &\leftrightarrow V_t^\ell \not\subseteq X \text{ for all } \ell \in L_t^{live},\\ \mathbf{1}_L \in f(\ell) &\leftrightarrow X_L \cap V_t^\ell \neq \emptyset \text{ for all } \ell \in L_t^{live},\\ \mathbf{1}_R \in f(\ell) &\leftrightarrow X_R \cap V_t^\ell \neq \emptyset \text{ for all } \ell \in L_t^{live}\}.\end{aligned}$$

Instead of computing the sets $\mathcal{A}_t^{\overline{c},\overline{w}}(f)$ directly, we compute only the parity of their cardinality, i.e., $A_t^{\overline{c},\overline{w}}(f) = |\mathcal{A}_t^{\overline{c},\overline{w}}(f)| \mod 2$.

We can now argue more formally that the exclusion of the states $\emptyset$ and Ω does not cause issues. First, for any nonempty V_t^ℓ at least one of the three cases $V_t^\ell \not\subseteq X$, $X_L \cap V_t^\ell \neq \emptyset$, or $X_R \cap V_t^\ell \neq \emptyset$ has to occur, hence the state $\emptyset$ cannot be attained by any V_t^ℓ with $\ell \in L_t^{live}$. Secondly, consider some node t that is not the child of a dead node, and $(X,(X_L,X_R)) \in \mathcal{A}_t^{\overline{c},\overline{w}}$ such that there is some live label $\ell \in L_t^{live}$ for which the three cases $V_t^\ell \not\subseteq X$, $X_L \cap V_t^\ell \neq \emptyset$, and $X_R \cap V_t^\ell \neq \emptyset$ simultaneously occur. Since ℓ is a live label, there is some $v \in N_G(V_t^\ell) \setminus N_{G_t}(V_t^\ell)$ by Lemma 5.1.8. We claim that $(X,(X_L,X_R))$ cannot be extended to a consistently cut vertex cover $(X',(X'_L,X'_R))$ of $G' = G[V_t \cup \{v\}]$ (and hence also not of G). If $v \notin X'$, then there is an uncovered edge in G' between V_t^ℓ and v. If $v \in X'$, then there is an edge in G' crossing the cut (X'_L, X'_R) and so the cut cannot be consistent. Hence, we can safely discard any partial solutions that attain the state $\{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$ with a live label, as they can never be extended to a global solution.

The state $\Omega = \{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$ can be obtained when two sets of vertices become united under a common label, i.e., by a relabel-operation or union-operation. We give recurrences for the quantities $\mathcal{A}_t^{\overline{c},\overline{w}}(f)$, where f is a t-signature which cannot attain Ω for any label, hence such situations are implicitly filtered out in the algorithm as the recurrences simply do not consider state combinations that lead to Ω.

We proceed to give recurrences for computing $A_t^{\overline{c},\overline{w}}(f)$, for every $t \in V(\hat{T}_\mu)$, t-signature f, $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$ depending on the type of the node t.

Introduce Node. If $t = \ell(v)$ for some $\ell \in [k]$, then $L_t^{live} = \{\ell\}$ and

$$\begin{aligned}A_t^{\overline{c},\overline{w}}(f) = {}&[v \neq v_* \vee f(\ell) = \{\mathbf{1}_L\}]\\ &\cdot [(f(\ell) = \{\mathbf{0}\} \wedge \overline{c} = \overline{w} = 0) \vee (f(\ell) \in \{\{\mathbf{1}_L\},\{\mathbf{1}_R\}\} \wedge \overline{c} = \mathbf{c}(v) \wedge \overline{w} = \mathbf{w}(v))],\end{aligned}$$

since in a singleton graph any choice of singleton state leads to a valid solution, but if $v = v_*$ then only the solution with v_* on the left side of the cut is allowed.

Relabel Node. If $t = \rho_{i\to j}(G_{t'})$, where t' is the child of t, for some $i, j \in [k]$, then $i \in L_{t'}$, $j \in L_{t'}$, $L_t = L_{t'} \setminus \{i\}$ and either $i, j \in L_{t'}^{live}$ or $i, j \in L_{t'}^{dead}$, since μ is nice.

- If labels i and j are live at t', then j is live at t and the recurrence is given by

$$A_t^{\overline{c},\overline{w}}(f) = \sum_{\substack{\mathbf{S}_1,\mathbf{S}_2 \in \mathbf{States}:\\ \mathbf{S}_1 \cup \mathbf{S}_2 = f(j)}} A_{t'}^{\overline{c},\overline{w}}(f[i \mapsto \mathbf{S}_1, j \mapsto \mathbf{S}_2]),$$

since $V_t^j = V_{t'}^i \cup V_{t'}^j$ and we simply have to iterate over all possible combinations of previous states at labels i and j that yield the desired state $f(j)$.

- If labels i and j are dead at t', then label j is dead at t and since we do not track the state of dead labels, we can simply copy the previous table, i.e.,

$$A_t^{\overline{c},\overline{w}}(f) = A_{t'}^{\overline{c},\overline{w}}(f).$$

Join Node. To check whether two states can lead to a feasible solution after adding a join between their labels, we introduce a helper function $\mathrm{feas}\colon \mathbf{States} \times \mathbf{States} \to \{0,1\}$ defined by $\mathrm{feas}(\mathbf{S}_1, \mathbf{S}_2) = [\mathbf{0} \notin \mathbf{S}_1 \vee \mathbf{0} \notin \mathbf{S}_2][\mathbf{1}_L \in \mathbf{S}_1 \to \mathbf{1}_R \notin \mathbf{S}_2][\mathbf{1}_R \in \mathbf{S}_1 \to \mathbf{1}_L \notin \mathbf{S}_2]$, or equivalently by the following table:

feas	$\{\mathbf{0}\}$	$\{\mathbf{1}_L\}$	$\{\mathbf{1}_R\}$	$\{\mathbf{0},\mathbf{1}_L\}$	$\{\mathbf{0},\mathbf{1}_R\}$	$\{\mathbf{1}_L,\mathbf{1}_R\}$
$\{\mathbf{0}\}$	0	1	1	0	0	1
$\{\mathbf{1}_L\}$	1	1	0	1	0	0
$\{\mathbf{1}_R\}$	1	0	1	0	1	0
$\{\mathbf{0},\mathbf{1}_L\}$	0	1	0	0	0	0
$\{\mathbf{0},\mathbf{1}_R\}$	0	0	1	0	0	0
$\{\mathbf{1}_L,\mathbf{1}_R\}$	1	0	0	0	0	0

There are two reasons for infeasibility: a join edge is not covered, i.e., $\mathbf{0}$ appears on both sides, or a join edge connects both sides of the cut, i.e., $\mathbf{1}_L$ appears on one side and $\mathbf{1}_R$ on the other. Using this helper function, we can now state the recurrence.

We have that $t = \eta_{i,j}(G_{t'})$ for some $i \neq j \in L_{t'}$ and where t' is the child of t. We must have $i, j \in L_{t'}^{live}$ and if the set of dead vertices changes, i.e., $D_t \neq D_{t'}$, then this will be handled by future dead nodes. Hence, we simply have to filter out all partial solutions that became infeasible due to the new join:

$$A_t^{\overline{c},\overline{w}}(f) = \mathrm{feas}(f(i), f(j)) A_{t'}^{\overline{c},\overline{w}}(f).$$

Dead Node. We have that $t = \bot_\ell(G_{t'})$, where t' is the child of t, $\ell \notin L_t^{live}$, and $L_t^{live} = L_{t'}^{live} \setminus \{\ell\}$. Since the only change is that t-signatures do not track the state of label ℓ anymore, we have to add up the contributions of all previous states of label ℓ. Hence, the recurrence is given by

$$A_t^{\overline{c},\overline{w}}(f) = \sum_{\mathbf{S} \in \mathbf{States}} A_{t'}^{\overline{c},\overline{w}}(f[\ell \mapsto \mathbf{S}]).$$

Since $\mathbf{States} \neq \mathcal{P}(\Omega)$, this recurrence could a priori fail to account for some partial solutions, e.g., those where all three vertex states simultaneously appear at label ℓ, however the previous argument on the exclusion of label state Ω essentially shows that such partial solutions cannot exist at this dead node. So, although the recurrence looks simple, its correctness proof is nontrivial.

Union Node. We have that $t = G_{t_1} \oplus G_{t_2}$, where t_1 and t_2 are the children of t and we have $L_t^{live} = L_{t_1}^{live} \cup L_{t_2}^{live}$. Given a t-signature f, we consider the union-split $f_{t,1}$, $f_{t,2}$, $f_{t,12}$ of f at t, cf. Definition 5.1.11. For every label $\ell \in \tilde{L}_{t,12} = L_{t_1}^{live} \cap L_{t_2}^{live}$, we need to consider

all states $\mathbf{S}_1, \mathbf{S}_2 \in \mathbf{States}$ such that $\mathbf{S}_1 \cup \mathbf{S}_2 = f(\ell)$, where $\mathbf{S}_i$ is the state of label ℓ at t_i. Furthermore, we have to distribute $\bar{c}$ and $\overline{w}$ among the partial solutions at t_1 and the partial solutions at t_2. Hence, we obtain the recurrence

$$A_t^{\bar{c},\overline{w}}(f) = \sum_{\substack{\bar{c}_1+\bar{c}_2=\bar{c} \\ \overline{w}_1+\overline{w}_2=\overline{w}}} \sum_{\substack{g_1,g_2\colon \tilde{L}_{t,12}\to\mathbf{States}\colon \\ g_1(\ell)\cup g_2(\ell)=f_{t,12}(\ell)\ \forall \ell\in\tilde{L}_{t,12}}} A_{t_1}^{\bar{c}_1,\overline{w}_1}(g_1 \cup f_{t,1}) A_{t_2}^{\bar{c}_2,\overline{w}_2}(g_2 \cup f_{t,2}),$$

where we interpret g_i and $f_{t,i}$ as sets in the expression $g_i \cup f_{t,i}$ for $i \in [2]$.

We compute this recurrence in time $\mathcal{O}^*(6^{|L_t^{live}|})$ for fixed $\bar{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$, and for all t-signatures f simultaneously as follows. We branch on all possibilities for $(\bar{c}_1, \bar{c}_2, \overline{w}_1, \overline{w}_2)$; these are $n^{\mathcal{O}(1)}$ possibilities as $\mathbf{c}(V) \leq n^{\mathcal{O}(1)}$ and $\mathbf{w}(V) \leq 2n^2$. Fixing one of these possibilities, we branch on $f_{t,1}$ and $f_{t,2}$, this leads to $6^{|\tilde{L}_{t,1}|+|\tilde{L}_{t,2}|}$ choices. The quantities $A_{t_1}^{\bar{c}_1,\overline{w}_1}(g_1 \cup f_{t,1})$ and $A_{t_2}^{\bar{c}_2,\overline{w}_2}(g_2 \cup f_{t,2})$ can be considered as functions of g_1 and g_2 and the inner sum over these in the recurrence is their componentwise cover product over $\mathbf{States}$ evaluated at $f_{t,12}$.

Since the set $\mathbf{States}$ is clearly a closure difference, we can apply Theorem 5.2.6 to compute this componentwise cover product in time $\mathcal{O}^*(6^{|\tilde{L}_{t,12}|})$ for all possible $f_{t,12}$. Since L_t^{live} is partitioned into $\tilde{L}_{t,1}$, $\tilde{L}_{t,2}$, and $\tilde{L}_{t,12}$ by Lemma 5.1.12, we need in total time $\mathcal{O}^*(6^{|L_t^{live}|})$ to compute $A_t^{\bar{c},\overline{w}}(f)$ for all choices of $\bar{c}$, $\overline{w}$, and f.

Lemma 5.3.1. *Given a nice k-expression μ of $G = (V, E)$, there is an algorithm that computes the quantities $A_t^{\bar{c},\overline{w}}(f)$ for all nodes $t \in V(\hat{T}_\mu)$, all t-signatures f, and all $\bar{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, 2n^2]$, in time $\mathcal{O}^*(6^k)$.*

Proof. The algorithm proceeds by bottom-up dynamic programming along the augmented syntax tree $\hat{T}_\mu$ of the nice k-expression μ and computes the quantities $A_t^{\bar{c},\overline{w}}(f)$ via the given recurrences. For an introduce node, relabel node, or join node, the recurrence for $A_t^{\bar{c},\overline{w}}(f)$ for fixed f, $\bar{c}$, and $\overline{w}$, can clearly be computed in polynomial time, since additions and multiplications in $\mathbb{Z}_2$ take constant time. For a union node t, we have argued how to compute the recurrences for all f, $\bar{c}$, and $\overline{w}$ simultaneously in time $\mathcal{O}^*(6^{|L_t^{live}|})$. As μ is a k-expression, we have $|L_t^{live}| \leq k$ for all $t \in V(\hat{T}_\mu)$ and in particular at most 6^k t-signatures for any node $t \in V(T_\mu)$. Hence, the running time follows as μ consists of a polynomial number of operations.

It remains to prove the correctness of the recurrences. The proof of correctness for introduce nodes, relabel nodes, join nodes, and union nodes is straightforward and hence omitted. Consider a dead node $t = \bot_\ell(G_{t'})$, where t' is the child of t. We claim that $\mathcal{A}_t^{\bar{c},\overline{w}}(f) = \bigcup_{\mathbf{S}\in\mathbf{States}} \mathcal{A}_{t'}^{\bar{c},\overline{w}}(f[\ell \mapsto \mathbf{S}])$ for all t-signatures f, $\bar{c}$, and $\overline{w}$. Since the union on the right-hand side is clearly disjoint, this claim immediately proves the correctness of the recurrence for dead nodes. We proceed with proving the claim. The right-hand side is contained in the left-hand side, since the set on the left-hand side is defined by fewer constraints.

For the other direction, suppose there is some partial solution $(X, (X_L, X_R)) \in \mathcal{A}_t^{\bar{c},\overline{w}}(f) \setminus \left(\bigcup_{\mathbf{S}\in\mathbf{States}} \mathcal{A}_{t'}^{\bar{c},\overline{w}}(f[\ell \mapsto \mathbf{S}])\right)$ and consider V_t^ℓ. Since $V_t^\ell \neq \emptyset$, at least one of $V_t^\ell \not\subseteq X$, $X_L \cap V_t^\ell \neq \emptyset$, $X_R \cap V_t^\ell \neq \emptyset$ has to be satisfied. Indeed, all three statements are satisfied simultaneously, because all remaining cases are covered by $\mathbf{States}$. Consider the join node

t^* that caused the dead node t to exist. The node t^* adds the final join incident to $V_t^\ell = V_{t^*}^\ell$, say between V_t^ℓ and some $V_t^i = V_{t^*}^i$, $i \neq \ell$, and by niceness of μ we have $V_t^i \neq \emptyset$. Hence, at least one of $V_t^i \not\subseteq X$, $X_L \cap V_t^i \neq \emptyset$, $X_R \cap V_t^i \neq \emptyset$ has to be satisfied. Therefore, there is an uncovered edge between V_t^ℓ and V_t^i or an edge crossing the cut (X_L, X_R), contradicting that $(X, (X_L, X_R)) \in \mathcal{A}_t^{\overline{c},\overline{w}}(f)$. Hence, $\mathcal{A}_t^{\overline{c},\overline{w}}(f) \subseteq \bigcup_{\mathbf{S} \in \mathbf{States}} \mathcal{A}_{t'}^{\overline{c},\overline{w}}(f[\ell \mapsto \mathbf{S}])$, proving the claim. □

Theorem 5.3.2. *There is a randomized algorithm that given a nice k-expression μ for a graph $G = (V, E)$ can solve* Connected Vertex Cover *in time $\mathcal{O}^*(6^k)$. The algorithm does not return false positives and returns false negatives with probability at most $1/2$.*

Proof. We begin by sampling a weight function $\mathbf{w} \colon V \to [2n]$ uniformly at random. Then, we pick an edge in G and branch on its endpoints; the chosen endpoint takes the role of v_* in the current branch. We then run the algorithm of Lemma 5.3.1 to compute the quantities $A_t^{\overline{c},\overline{w}}(f)$. Let $\hat{r}$ denote the root node of the expression μ. At the root, we have that $L_{\hat{r}}^{live} = \emptyset$. The algorithm returns true if in one of the branches there is some choice of $\overline{c} \leq \overline{b}$, $\overline{w} \in [0, 2n^2]$, such that $A_{\hat{r}}^{\overline{c},\overline{w}}(\emptyset) \neq 0$, otherwise the algorithm returns false.

The running time directly follows from Lemma 5.3.1. For the correctness, first note that at the root, we have $\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}(\emptyset) = \mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}$. The algorithm only returns true, if there are some $\overline{c} \in [0, \overline{b}]$, $\overline{w} \in [0, 2n^2]$ such that $\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}$ has odd cardinality. By Theorem 3.1.6, this implies that there is a connected vertex cover X of G with $\mathbf{c}(X) = \overline{c} \leq \overline{b}$, hence the algorithm does not return false positives.

For the error probability, suppose that the weight function $\mathbf{w}$ isolates an optimum connected vertex cover X^* of G and that $\mathbf{c}(X^*) \leq \overline{b}$; by Lemma 3.1.4, the isolation happens with probability greater than or equal to $1/2$. Furthermore, consider a branch with $v_* \in X^*$. Set $\overline{c} = \mathbf{c}(X^*) \leq \overline{b}$ and $\overline{w} = \mathbf{w}(X^*)$. By Lemma 3.1.1, the connected vertex cover X^* contributes once to $|\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}|$ and all other contributing sets cannot be connected due to isolation and hence contribute an even number to $|\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}|$. Therefore $A_{\hat{r}}^{\overline{c},\overline{w}}(\emptyset) \equiv_2 |\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}| = 1 \neq 0$ and hence the algorithm returns true with probability at least $1/2$ given a positive instance. □

5.4 Connected Dominating Set Algorithm

When solving Connected Dominating Set, we only consider instances where the costs are polynomially bounded in the input size. Furthermore, we assume without loss of generality that G is connected and contains at least two vertices.

We begin by motivating our algorithmic approach for Connected Dominating Set. Following the approach for Connected Vertex Cover, we would consider partial solutions $(X, (X_L, X_R))$ consisting of a *partial* dominating set X and a consistent cut (X_L, X_R) of the subgraph induced by X. A single vertex v can take four states with respect to $(X, (X_L, X_R))$: $\mathbf{0}_1$, $\mathbf{0}_0$, $\mathbf{1}_L$, $\mathbf{1}_R$, where the former two indicate that $v \notin X$ and the subscript denotes whether v is dominated by X or not, and the latter two indicate that $v \in X$ and the subscript denotes which cut side contains v. We can again store for each label which vertex states appear, yielding as possible label states all subsets of $\{\mathbf{0}_1, \mathbf{0}_0, \mathbf{1}_L, \mathbf{1}_R\}$. By observing that the state $\mathbf{0}_1$ does not impose any constraint for future joins, we can even argue that it suffices to only

consider the subsets of $\{\mathbf{0}_0, \mathbf{1}_L, \mathbf{1}_R\}$. Furthermore, similar to Connected Vertex Cover, the label state $\{\mathbf{0}_0, \mathbf{1}_L, \mathbf{1}_R\}$ cannot be sensibly attained by live labels, hence we are down to seven states per live label. This approach yields a running time of $\mathcal{O}^*(7^{\mathrm{cw}(G)})$, but we can obtain an even faster algorithm.

To obtain the improved running time of $\mathcal{O}^*(5^{\mathrm{cw}(G)})$, we instead work with a different set of vertex states common for domination problems [140, 155, 164] and which will appear again in Part III. Instead of considering partial solutions with *dominated* and *undominated* vertices, we consider *allowed* vertices (state $\mathbf{A}$) and *forbidden* vertices (state $\mathbf{F}$). As their names imply, allowed vertices may be dominated or undominated, but forbidden vertices may not be dominated. When lifting the *vertex* states to *label* states, the state $\mathbf{A}$ can be ignored, because it imposes no constraint on joins, therefore we obtain the subsets of $\{\mathbf{F}, \mathbf{L}, \mathbf{R}\}$ as *label states*. The advantage of this set of states is that all subsets of size at least two behave the same with respect to joins. This allows us to collapse them to a single state and ensures that we do not have to update states from undominated to dominated when handling joins.

However, recovering the solutions to the original problem from this set of states usually requires some type of inclusion-exclusion argument. The application of this step is non-standard for clique-width. For sparse graph parameters, such as treewidth, the inclusion-exclusion argument can be applied to *single* vertices, i.e., if we subtract the partial solutions where a vertex v has state $\mathbf{F}$ from those where v has state $\mathbf{A}$, then only partial solutions dominating v remain. For clique-width however, we have to apply the argument to groups of vertices and such a subtraction would only yield that *some* vertices in the label must be dominated and not, as is desired, all of them. Moreover, the collapsing of several label states into a single one complicates the inclusion-exclusion argument further. Surprisingly, working modulo 2 resolves all of these problems simultaneously and it is also a natural setting for the cut-and-count-technique. Lastly, the inclusion-exclusion argument should only be applied when all edges incident to a label are already constructed, i.e., the considered label is *dead*, hence we again use augmented syntax trees.

We proceed by giving the formal details of the algorithm. Given a k-expression μ for $G = (V, E)$, we can assume that μ is nice after polynomial-time preprocessing, see Lemma 5.1.4. We sample a weight function $\mathbf{w}\colon V \to [2n]$ for the isolation lemma, cf. Lemma 3.1.4. To solve Connected Dominating Set, we perform bottom-up dynamic programming along the augmented syntax tree $\hat{T}_\mu$ of μ. We pick some $v_* \in V$ and only consider solutions containing v_* on the left side of the cut. This allows us to work modulo 2 for the cut-and-count-technique which synergizes with the inclusion-exclusion approach in this case.

To implement the inclusion-exclusion and cut-and-count approach, we consider the following family of partial solutions at a node $t \in V(\hat{T}_\mu)$.

Definition 5.4.1. At a node $t \in V(\hat{T}_\mu)$, the family $\mathcal{A}_t$ of *partial solutions* consists of all ordered subpartitions (X_L, X_R, F) of V_t satisfying the following properties:

- $X = X_L \cup X_R$, (X_L, X_R) is a consistent cut of $G_t[X]$,
- $v_* \in X_L$ if $v_* \in V_t$,
- $N_{G_t}[X] \cap F = \emptyset$,
- $V_t^\ell \subseteq N_{G_t}[X]$ for all $\ell \in L_t^{dead}$.

Furthermore, for every node $t \in V(\hat{T}_\mu)$, $\overline{c} \in [0, \mathbf{c}(V)]$, and $\overline{w} \in [0, \mathbf{w}(V)]$, we define $\mathcal{A}_t^{\overline{c},\overline{w}} = \{(X_L, X_R, F) \in \mathcal{A}_t : \mathbf{c}(X_L \cup X_R) = \overline{c}, \mathbf{w}(X_L \cup X_R) = \overline{w}\}$.

Essentially, every $(X_L, X_R, F) \in \mathcal{A}_t$ consists of a consistently cut *partial* dominating set $X = X_L \cup X_R$ of G_t that dominates all vertices with dead labels and does *not* dominate the vertices in F. To any $(X_L, X_R, F) \in \mathcal{A}_t$, there is also an associated set of vertices $A = V_t \setminus (X_L \cup X_R \cup F)$ that are *allowed* to be dominated.

Notice that for any $(X_L, X_R, F) \in \mathcal{A}_t$, we must have that $F \cap \left(\bigcup_{\ell \in L_t^{dead}} V_t^\ell\right) = \emptyset$ as otherwise the third and fourth property in the definition of $\mathcal{A}_t$ cannot be simultaneously satisfied. In particular, for the root node $\hat{r}$ we must have $F = \emptyset$ and $L_{\hat{r}}^{dead} = L_{\hat{r}}$, so that $\mathcal{A}_{\hat{r}}$ only contains consistently cut dominating sets of G. Hence, if there are $\overline{c} \in [0, \overline{b}]$, $\overline{w} \in [0, 2n^2]$, such that $\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}$ has odd cardinality, then there exists a connected dominating set X of G with $\mathbf{c}(X) \leq \overline{b}$ by Theorem 3.1.6.

To compute the sets $\mathcal{A}_t^{\overline{c},\overline{w}}$ via dynamic programming, we partition the partial solutions according to their states on the live labels L_t^{live}. The state of a partial solution $(X_L, X_R, F) \in \mathcal{A}_t$ at a label $\ell \in L_t^{live}$ is based on which of the sets X_L, X_R, and F are intersected by V_t^ℓ. To capture this, we make the following definition.

Definition 5.4.2. Let $t \in V(\hat{T}_\mu)$ be a node of $\hat{T}_\mu$ and $\ell \in L_t$ be a nonempty label. Given an ordered subpartition (X_L, X_R, F) of V_t, we define the set $\mathbf{S}_t^\ell(X_L, X_R, F) \subseteq \{\mathbf{F}, \mathbf{L}, \mathbf{R}\}$ by

- $\mathbf{F} \in \mathbf{S}_t^\ell(X_L, X_R, F) \iff F \cap V_t^\ell \neq \emptyset$,
- $\mathbf{L} \in \mathbf{S}_t^\ell(X_L, X_R, F) \iff X_L \cap V_t^\ell \neq \emptyset$,
- $\mathbf{R} \in \mathbf{S}_t^\ell(X_L, X_R, F) \iff X_R \cap V_t^\ell \neq \emptyset$.

Naively, using the sets $\mathbf{S}_t^\ell(X_L, X_R, F)$ would yield $2^3 = 8$ states per label. Note that also $\mathbf{S}_t^\ell(X_L, X_R, F) = \emptyset$ is sensible for $(X_L, X_R, F) \in \mathcal{A}_t$ and nonempty label $\ell \in L_t$, because this simply means $V_t^\ell \subseteq A = V_t \setminus (X_L \cup X_R \cup F)$, i.e., all vertices in V_t^ℓ are allowed to be dominated and not part of the partial dominating set $X_L \cup X_R$. Surprisingly, it turns out that all $\mathbf{S} \subseteq \{\mathbf{F}, \mathbf{L}, \mathbf{R}\}$ with $|\mathbf{S}| \geq 2$ can be handled in the same way, since all such $\mathbf{S}$ can only be feasibly joined to the state $\emptyset$. This enables us to solve Connected Dominating Set with 5 states per label instead of 8. We define the set $\mathbf{States} = \{\emptyset, \{\mathbf{F}\}, \{\mathbf{L}\}, \{\mathbf{R}\}, \mathbf{2}_+\}$, where $\mathbf{2}_+$ is not a subset of $\{\mathbf{F}, \mathbf{L}, \mathbf{R}\}$ but a formal symbol representing the subsets of size at least 2.

Definition 5.4.3. Given a node $t \in V(\hat{T}_\mu)$, a *t-signature* is a function $f \colon L_t^{live} \to \mathbf{States}$. A subpartition (X_L, X_R, F) of V_t is *compatible* with a t-signature f if for all $\ell \in L_t^{live}$ the following two properties hold:

- $\mathbf{S}_t^\ell(X_L, X_R, F) = f(\ell)$ if $f(\ell) \neq \mathbf{2}_+$,
- $|\mathbf{S}_t^\ell(X_L, X_R, F)| \geq 2$ if $f(\ell) = \mathbf{2}_+$.

Furthermore, for every node $t \in V(\hat{T}_\mu)$, t-signature f, $\overline{c} \in [0, \mathbf{c}(V)]$, and $\overline{w} \in [0, \mathbf{w}(V)]$, we define $\mathcal{A}_t^{\overline{c},\overline{w}}(f) = \{(X_L, X_R, F) \in \mathcal{A}_t^{\overline{c},\overline{w}} : (X_L, X_R, F)$ is compatible with $f\}$.

Unlike Connected Vertex Cover, there are no states in Connected Dominating Set that can only appear at dead labels, since the state $\emptyset$ can be joined to any state without making the partial solution infeasible. Regardless, distinguishing live and dead labels remains useful, as we only want to require the domination of vertices with dead labels; vertices with live labels may be dominated by an edge that is missing at the current node. Hence, the dead nodes of $\hat{T}_\mu$ serve as natural nodes to apply the inclusion-exclusion step.

As usual, we do not compute the sets $\mathcal{A}_t^{\overline{c},\overline{w}}(f)$ directly, but the parity of their cardinality, i.e., $A_t^{\overline{c},\overline{w}}(f) = |\mathcal{A}_t^{\overline{c},\overline{w}}(f)| \mod 2$. We now proceed by presenting the various recurrences for $A_t^{\overline{c},\overline{w}}(f)$, given node $t \in V(\hat{T}_\mu)$, t-signature f, $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$, based on the type of the node t.

Introduce Node. If $t = \ell(v)$ for some $\ell \in [k]$ and $v \in V$, then $L_t^{live} = \{\ell\}$ as v cannot be an isolated vertex by assumption and

$$A_t^{\overline{c},\overline{w}}(f) = [v \neq v_* \vee f(\ell) = \{\mathbf{L}\}] \cdot [(\overline{c} = \overline{w} = 0 \wedge f(\ell) \in \{\emptyset, \{\mathbf{F}\}\}) \vee (\overline{c} = \mathbf{c}(v) \wedge \overline{w} = \mathbf{w}(v) \wedge f(\ell) \in \{\{\mathbf{L}\}, \{\mathbf{R}\}\})],$$

first checking for the edge case that $v = v_*$ and then that $\overline{c}$ and $\overline{w}$ agree with the chosen state. Note that the state $\mathbf{2}_+$ cannot be achieved here.

Relabel Node. If $t = \rho_{i\to j}(G_{t'})$, where t' is the only child of t, then label i and j are nonempty at node t', since μ is a nice expression. By irredundancy of the expression μ, either $\{i,j\} \subseteq L_{t'}^{dead}$ or $\{i,j\} \cap L_{t'}^{dead} = \emptyset$. In the first case, we have that $j \in L_t^{dead}$ and the recurrence is simply $A_t^{\overline{c},\overline{w}}(f) = A_{t'}^{\overline{c},\overline{w}}(f)$, since we do not store the state of dead labels and the domination requirement for dead labels remains satisfied.

In the second case, we have $j \in L_t^{live}$ and we iterate through all pairs of states that combine to the current state at label j. Since $V_t^j = V_{t'}^i \cup V_{t'}^j$, the recurrence is

$$A_t^{\overline{c},\overline{w}}(f) = \sum_{\substack{\mathbf{S}_1,\mathbf{S}_2 \in \mathbf{States}: \\ \mathrm{merge}(\mathbf{S}_1,\mathbf{S}_2) = f(j)}} A_{t'}^{\overline{c},\overline{w}}(f[i \mapsto \mathbf{S}_1, j \mapsto \mathbf{S}_2]),$$

where $\mathrm{merge}\colon \mathbf{States} \times \mathbf{States} \to \mathbf{States}$ is given by:

merge	$\emptyset$	$\{\mathbf{F}\}$	$\{\mathbf{L}\}$	$\{\mathbf{R}\}$	$\mathbf{2}_+$
$\emptyset$	$\emptyset$	$\{\mathbf{F}\}$	$\{\mathbf{L}\}$	$\{\mathbf{R}\}$	$\mathbf{2}_+$
$\{\mathbf{F}\}$	$\{\mathbf{F}\}$	$\{\mathbf{F}\}$	$\mathbf{2}_+$	$\mathbf{2}_+$	$\mathbf{2}_+$
$\{\mathbf{L}\}$	$\{\mathbf{L}\}$	$\mathbf{2}_+$	$\{\mathbf{L}\}$	$\mathbf{2}_+$	$\mathbf{2}_+$
$\{\mathbf{R}\}$	$\{\mathbf{R}\}$	$\mathbf{2}_+$	$\mathbf{2}_+$	$\{\mathbf{R}\}$	$\mathbf{2}_+$
$\mathbf{2}_+$	$\mathbf{2}_+$	$\mathbf{2}_+$	$\mathbf{2}_+$	$\mathbf{2}_+$	$\mathbf{2}_+$

Note that $\mathbf{S}_t^j(X_L, X_R, F) = \mathbf{S}_{t'}^i(X_L, X_R, F) \cup \mathbf{S}_{t'}^j(X_L, X_R, F)$ and if label i had state $\mathbf{S}_1$ at t' and label j state $\mathbf{S}_2$ at t', then label j has state $\mathrm{merge}(\mathbf{S}_1, \mathbf{S}_2)$ at node t.

Join Node. If $t = \eta_{i,j}(G_{t'})$, $i \neq j$, where t' is the only child of t, then we know that $i, j \in L_{t'}^{live}$ since the expression μ is nice. Due to this join, the vertices with label i or j could receive their final incident edges, possibly leading to $V_t^i \subseteq D_t$ or $V_t^j \subseteq D_t$. If this happens, this join will be followed by up to two dead nodes. At this join, we filter out the partial solutions that are invalidated by the newly added edges, hence the recurrence is given by

$$A_t^{\overline{c},\overline{w}}(f) = \mathrm{feas}(f(i), f(j)) A_{t'}^{\overline{c},\overline{w}}(f),$$

where $\mathrm{feas}\colon \mathbf{States} \times \mathbf{States} \to \{0,1\}$ is given by the following table:

feas	$\emptyset$	$\{\mathbf{F}\}$	$\{\mathbf{L}\}$	$\{\mathbf{R}\}$	$\mathbf{2}_+$
$\emptyset$	1	1	1	1	1
$\{\mathbf{F}\}$	1	1	0	0	0
$\{\mathbf{L}\}$	1	0	1	0	0
$\{\mathbf{R}\}$	1	0	0	1	0
$\mathbf{2}_+$	1	0	0	0	0

There are two reasons why a partial solution (X_L, X_R, F) might be invalidated; a new edge connects F and $X = X_L \cup X_R$ or a new edge connects X_L and X_R.

Dead Node. Suppose that $t = \bot_\ell(G_{t'})$, where $\ell \in L'_t$ and t' is the child of t. We have that $L_t^{live} = L_{t'}^{live} \setminus \{\ell\}$ and $\ell \in L_t^{dead}$. Due to the definition of $\mathcal{A}_t$, we only want to count the partial solutions from $\mathcal{A}_{t'}$ that dominate V_t^ℓ completely. If V_t^ℓ contains only a single vertex, then this is easy to check with the given states: we could simply compute for a t-signature f

$$A_{t'}^{\bar{c},\overline{w}}(f[\ell \mapsto \{\mathbf{L}\}]) + A_{t'}^{\bar{c},\overline{w}}(f[\ell \mapsto \{\mathbf{R}\}]) + (A_{t'}^{\bar{c},\overline{w}}(f[\ell \mapsto \emptyset]) - A_{t'}^{\bar{c},\overline{w}}(f[\ell \mapsto \{\mathbf{F}\}])),$$

where the last part is the inclusion-exclusion argument checking that the vertex in V_t^ℓ is dominated. However, if $|V_t^\ell| \geq 2$ then we also need to handle the state $\mathbf{2}_+$ for which it is unclear how to incorporate it into an inclusion-exclusion argument as we do not know the value of $\mathbf{S}_{t'}^\ell(X_L, X_R, F)$ in this case. Furthermore, the previous inclusion-exclusion argument is invalid over $\mathbb{Z}$ if $|V_t^\ell| \geq 2$ as partial solutions with multiple undominated vertices in V_t^ℓ are counted several times by $A_{t'}^{\bar{c},\overline{w}}(f[\ell \mapsto \{\mathbf{F}\}])$.

Surprisingly, there is a very simple recurrence that avoids all these issues modulo 2. If f is a t-signature, and $\bar{c} \in [0, n]$, $\overline{w} \in [0, 2n^2]$, then the recurrence is given by

$$A_t^{\bar{c},\overline{w}}(f) = \sum_{\mathbf{S} \in \mathbf{States}} A_{t'}^{\bar{c},\overline{w}}(f[\ell \mapsto \mathbf{S}]).$$

This recurrence works because any partial dominating set containing exactly u undominated vertices in V_t^ℓ is counted 2^u times by the right-hand side of the recurrence and hence cancels modulo 2 for $u > 0$. We proceed by giving the formal proof of correctness for this recurrence.

Proof. Fix f, $\bar{c}$, and $\overline{w}$. The left side counts the cardinality of $\mathcal{A}_t^{\bar{c},\overline{w}}(f)$ modulo 2 and the right side clearly computes $\sum_{\mathbf{S} \in \mathbf{States}} |\mathcal{A}_{t'}^{\bar{c},\overline{w}}(f[\ell \mapsto \mathbf{S}])|$ modulo 2. We have to prove that these terms agree. For readability, set $\mathcal{A}_{LHS} := \mathcal{A}_t^{\bar{c},\overline{w}}(f)$ and $\mathcal{A}_{RHS} := \bigcup_{\mathbf{S} \in \mathbf{States}} \mathcal{A}_{t'}^{\bar{c},\overline{w}}(f[\ell \mapsto \mathbf{S}])$.

Recall that $G_t = G_{t'}$, $V_t^i = V_{t'}^i$ for all $i \in L_t$, and $L_t^{dead} = L_{t'}^{dead} \cup \{\ell\}$ with $\ell \notin L_{t'}^{dead}$. First, notice that $\mathcal{A}_{LHS} \subseteq \mathcal{A}_{RHS}$, since for every possibility of $\mathbf{S}_{t'}^\ell(X_L, X_R, F)$, there is an $\mathbf{S} \in \mathbf{States}$ such that (X_L, X_R, F) is compatible with $f[\ell \mapsto \mathbf{S}]$ at node t'. Also note that the sets $\mathcal{A}_{t'}^{\bar{c},\overline{w}}(f[\ell \mapsto \mathbf{S}])$ are disjoint for distinct $\mathbf{S} \in \mathbf{States}$ and hence every element of $\mathcal{A}_{RHS}$ is counted exactly once on the right side of the equation.

Next, we see that for any $(X_L, X_R, F) \in \mathcal{A}_{RHS}$ that $(X_L, X_R, F) \in \mathcal{A}_{LHS}$ holds if and only if $V_t^\ell \subseteq N_{G_t}[X]$, where $X = X_L \cup X_R$, since the only requirements that change are the

compatibility with f, which is easier to satisfy at t than at t', and the requirement that all dead labels are dominated which requires $V_t^\ell \subseteq N_{G_t}[X]$ in addition to $(X_L, X_R, F) \in \mathcal{A}_{RHS}$.

It remains to show that $\mathcal{A}_{RHS} \setminus \mathcal{A}_{LHS}$ contains an even number of elements and hence cancels modulo 2. Consider $(X_L, X_R, F) \in \mathcal{A}_{RHS} \setminus \mathcal{A}_{LHS}$, let $X = X_L \cup X_R$ and let $U_t^\ell := U_t^\ell(X_L, X_R) := V_t^\ell \setminus N_{G_t}[X]$ be the set of undominated vertices with label ℓ at node t. From $(X_L, X_R, F) \in \mathcal{A}_{RHS} \setminus \mathcal{A}_{LHS}$ it follows that $U_t^\ell \neq \emptyset$, as otherwise $V_t^\ell \subseteq N_{G_t}[X]$ which contradicts $(X_L, X_R, F) \notin \mathcal{A}_{LHS}$ by the previous paragraph. For any $F' \subseteq U_t^\ell$, we see that $(X_L, X_R, (F \setminus V_t^\ell) \cup F') \in \mathcal{A}_{RHS} \setminus \mathcal{A}_{LHS}$, since U_t^ℓ remains unchanged but we only change which vertices are declared forbidden. Since $U_t^\ell \neq \emptyset$, there is an even number of choices, i.e., $2^{|U_t^\ell|}$ many, for F'.

Fixing some $\emptyset \neq U \subseteq V_t^\ell$, this shows there are an even number of elements $(X_L, X_R, F) \in \mathcal{A}_{RHS} \setminus \mathcal{A}_{LHS}$ with $U_t^\ell(X_L, X_R) = U$. Since every element of $\mathcal{A}_{RHS} \setminus \mathcal{A}_{LHS}$ is covered by some choice of U, it follows that the cardinality of $\mathcal{A}_{RHS} \setminus \mathcal{A}_{LHS}$ is even. □

Union Node. We have that $t = G_{t_1} \oplus G_{t_2}$, where t_1 and t_2 are the children of t and we have $L_t^{live} = L_{t_1}^{live} \cup L_{t_2}^{live}$. Given a t-signature f, we consider the union-split $f_{t,1}$, $f_{t,2}$, $f_{t,12}$ of f at t, cf. Definition 5.1.11. For every label $\ell \in \tilde{L}_{t,12} = L_{t_1}^{live} \cap L_{t_2}^{live}$, we need to consider all states $\mathbf{S}_1, \mathbf{S}_2 \in \mathbf{States}$ such that $\mathrm{merge}(\mathbf{S}_1, \mathbf{S}_2) = f(\ell)$, where $\mathbf{S}_i$ is the state of label ℓ at t_i. For two functions $g, h\colon B \to \mathbf{States}$, where B is some set, we write $\mathrm{merge}(g, h)\colon B \to \mathbf{States}$ for the *componentwise application* of merge, i.e. $\mathrm{merge}(g, h)(\ell) = \mathrm{merge}(g(\ell), h(\ell))$ for all $\ell \in B$. Furthermore, we distribute $\overline{c}$ and $\overline{w}$ among the partial solutions at t_1 and at t_2, leading to the recurrence

$$A_t^{\overline{c},\overline{w}}(f) = \sum_{\substack{\overline{c}_1+\overline{c}_2=\overline{c} \\ \overline{w}_1+\overline{w}_2=\overline{w}}} \sum_{\substack{g_1,g_2\colon \tilde{L}_{t,12}\to\mathbf{States}\colon \\ \mathrm{merge}(g_1,g_2)=f}} A_{t_1}^{\overline{c}_1,\overline{w}_1}(g_1 \cup f_{t,1}) A_{t_2}^{\overline{c}_2,\overline{w}_2}(g_2 \cup f_{t,2}),$$

where we consider g_i and $f_{t,i}$ as sets in $g_i \cup f_{t,i}$ for $i \in [2]$. Here, we simply fix the state for the labels that are live at only one child by using the parts $f_{t,1}$ and $f_{t,2}$ and for the labels that are live at both children we sum over all valid state combinations.

We now argue how this recurrence can be computed in time $\mathcal{O}^*(5^{|L_t^{live}|})$ for fixed $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$, and for all t-signatures f. First, we branch on the numbers $\overline{c}_1, \overline{c}_2 \in [0, \mathbf{c}(V)]$, $\overline{w}_1, \overline{w}_2 \in [0, \mathbf{w}(V)]$, and functions $h_1\colon \tilde{L}_{t,1} \to \mathbf{States}$, $h_2\colon \tilde{L}_{t,2} \to \mathbf{States}$. Having fixed these choices, we calculate the inner sum of the recurrence for all t-signatures f with $f_{t,1} = h_1$ and $f_{t,2} = h_2$, by setting $B_i(g) := A_{t_i}^{\overline{c}_i,\overline{w}_i}(g \cup h_i)$, for $i \in [2]$ and $g\colon \tilde{L}_{t,12} \to \mathbf{States}$, and computing the *CDS-product*

$$(B_1 \otimes_{CDS} B_2)(g) := \sum_{\substack{g_1,g_2\colon \tilde{L}_{t,12}\to\mathbf{States}\colon \\ \mathrm{merge}(g_1,g_2)=g}} B_1(g_1) B_2(g_2)$$

for all $g\colon \tilde{L}_{t,12} \to \mathbf{States}$. By the forthcoming Lemma 5.4.4, we can compute the CDS-product in time $\mathcal{O}^*(5^{|\tilde{L}_{t,12}|})$. Since there are $\mathcal{O}^*(5^{|\tilde{L}_{t,1}|+|\tilde{L}_{t,2}|})$ branches and L_t^{live} is partitioned into the three sets $\tilde{L}_{t,1}$, $\tilde{L}_{t,2}$, $\tilde{L}_{t,12}$ by Lemma 5.1.12, we need time $\mathcal{O}^*(5^{|L_t^{live}|})$ to compute the recurrence for all choices of $\overline{c}$, $\overline{w}$, and f.

Lemma 5.4.4. *Given two tables $B_1, B_2 \colon \mathbf{States}^I \to \mathbb{Z}_2$, where I is some index set, their CDS-product $B_1 \otimes_{CDS} B_2$ can be computed in time $\mathcal{O}^*(|\mathbf{States}|^{|I|}) = \mathcal{O}^*(5^{|I|})$.*

Proof. Consider the set family $\mathcal{L} = \{\emptyset, \{\mathbf{F}\}, \{\mathbf{L}\}, \{\mathbf{R}\}, \{\mathbf{F}, \mathbf{L}, \mathbf{R}\}\} \subseteq \mathcal{P}(U)$ over $U = \{\mathbf{F}, \mathbf{L}, \mathbf{R}\}$ and the partial order on $\mathcal{L}$ induced by set inclusion $\subseteq$. It is easy to verify that this forms a *lattice*, since every pair of elements has a greatest lower bound (*meet*/$\wedge$) and a least upper bound (*join*/$\vee$). In particular, the least upper bounds are given by:

$\vee$	$\emptyset$	$\{\mathbf{F}\}$	$\{\mathbf{L}\}$	$\{\mathbf{R}\}$	U
$\emptyset$	$\emptyset$	$\{\mathbf{F}\}$	$\{\mathbf{L}\}$	$\{\mathbf{R}\}$	U
$\{\mathbf{F}\}$	$\{\mathbf{F}\}$	$\{\mathbf{F}\}$	U	U	U
$\{\mathbf{L}\}$	$\{\mathbf{L}\}$	U	$\{\mathbf{L}\}$	U	U
$\{\mathbf{R}\}$	$\{\mathbf{R}\}$	U	U	$\{\mathbf{R}\}$	U
U	U	U	U	U	U

The bijection $\kappa \colon \mathbf{States} \to \mathcal{L}$, with $\kappa(\mathbf{S}) = \mathbf{S}$ for $\mathbf{S} \neq \mathbf{2}_+$ and $\kappa(\mathbf{2}_+) = U = \{\mathbf{F}, \mathbf{L}, \mathbf{R}\}$, turns merge on $\mathbf{States}$ into $\vee$ on $\mathcal{L}$, i.e., $\kappa(\mathrm{merge}(\mathbf{S}_1, \mathbf{S}_2)) = \kappa(\mathbf{S}_1) \vee \kappa(\mathbf{S}_2)$ for all $\mathbf{S}_1, \mathbf{S}_2 \in \mathbf{States}$. Hence, we can write

$$\begin{aligned}
(B_1 \otimes_{CDS} B_2)(g) &= \sum_{\substack{g_1, g_2 \colon I \to \mathbf{States} \colon \\ \mathrm{merge}(g_1, g_2) = g}} B_1(g_1) B_2(g_2) \\
&= \sum_{\substack{h_1, h_2 \colon I \to \mathcal{L} \colon \\ h_1(i) \vee h_2(i) = \kappa(g(i)) \; \forall i \in I}} B_1(\kappa^{-1} \circ h_1) B_2(\kappa^{-1} \circ h_2) \\
&= \sum_{\substack{h_1, h_2 \in \mathcal{L}^I \colon \\ h_1 \vee h_2 = \kappa \circ g}} B_1(\kappa^{-1} \circ h_1) B_2(\kappa^{-1} \circ h_2) = \sum_{\substack{h_1, h_2 \in \mathcal{L}^I \colon \\ h_1 \vee h_2 = \kappa \circ g}} B_1'(h_1) B_2'(h_2) \\
&= (B_1' \otimes_{\mathcal{L}^I} B_2')(\kappa \circ g),
\end{aligned}$$

where $B_j'(h) = B_j(\kappa^{-1} \circ h)$ for $h \in \mathcal{L}^I$, $j = 1, 2$, and $\vee$ is the join in $\mathcal{L}$ or $\mathcal{L}^I$ depending on the context. By identifying $\mathcal{L}^I$ with $\mathcal{L}^{|I|}$ in the natural way, we can therefore apply Corollary 5.2.10 to compute the CDS-product in time $\mathcal{O}^*(|\mathcal{L}|^{|I|}) = \mathcal{O}^*(5^{|I|})$ as the calls to $\mathbb{A}_{\mathcal{L}}$ can be answered in constant time. □

Lemma 5.4.5. *Given a nice k-expression μ of a graph G, there is an algorithm that computes the quantities $A_t^{\overline{c}, \overline{w}}(f)$ for all nodes $t \in V(T_\mu)$, all t-signatures f, and all $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, 2n^2]$, in time $\mathcal{O}^*(5^k)$.*

Proof. The algorithm proceeds by bottom-up dynamic programming along the augmented syntax tree $\hat{T}_\mu$ of the nice clique-expression μ and computes the quantities $A_t^{\overline{c}, \overline{w}}(f)$ via the given recurrences. For an introduce node, relabel node, join node, or dead node, the recurrence for $A_t^{\overline{c}, \overline{w}}(f)$ for fixed f, $\overline{c}$, and $\overline{w}$, can clearly be computed in polynomial time, since additions and multiplications in $\mathbb{Z}_2$ take constant time. For a union node t, we have argued how to compute the recurrences for all f, $\overline{c}$, and $\overline{w}$ simultaneously in time $\mathcal{O}^*(5^{|L_t^{live}|})$. As μ is a k-expression, we have $|L_t^{live}| \leq |L_t| \leq k$ for all $t \in V(T_\mu)$ and in particular at most 5^k t-signatures for any node $t \in V(T_\mu)$. Hence, the running time follows.

It remains to prove the correctness of the recurrences. We have already proven the correctness of the recurrence for the dead nodes. For the other node types, the proofs are straightforward.

For the join node, we highlight that the function $\mathrm{feas}\colon \mathbf{States} \times \mathbf{States} \to \{0,1\}$ satisfies $\mathrm{feas}(\mathbf{S}_1, \mathbf{S}_2) = 1 - [\mathbf{S}_1 \neq \emptyset][\mathbf{S}_2 \neq \emptyset][\mathbf{S}_1 = \mathbf{S}_2 = \mathbf{2}_+ \vee \mathbf{S}_1 \neq \mathbf{S}_2]$ which characterizes the state pairs where adding a join between their underlying labels results in an edge between $X = X_L \cup X_R$ and F or an edge across the cut (X_L, X_R), hence yielding an infeasible solution.

For relabel and union nodes, we highlight that for any two subsets $\mathbf{S}_1, \mathbf{S}_2 \subseteq \{\mathbf{F}, \mathbf{L}, \mathbf{R}\}$ we have $\rho(\mathbf{S}_1 \cup \mathbf{S}_2) = \mathrm{merge}(\rho(\mathbf{S}_1), \rho(\mathbf{S}_2))$, where $\rho\colon \mathcal{P}(\{\mathbf{F}, \mathbf{L}, \mathbf{R}\}) \to \mathbf{States}$ with $\rho(\mathbf{S}) = \mathbf{S}$ if $|\mathbf{S}| \leq 1$ and $\rho(\mathbf{S}) = \mathbf{2}_+$ if $|\mathbf{S}| \geq 2$. Hence, merge correctly updates the state for relabel and union nodes. □

Theorem 5.4.6. *There is a randomized algorithm that given a nice k-expression μ for a graph $G = (V, E)$ can solve* Connected Dominating Set *in time $\mathcal{O}^*(5^k)$. The algorithm does not return false positives and returns false negatives with probability at most $1/2$.*

Proof. We begin by sampling a weight function $\mathbf{w}\colon V \to [2n]$ uniformly at random. Then, we pick an arbitrary vertex v and branch on its closed neighborhood $N_G[v]$, since every dominating set intersects $N_G[v]$ in at least one vertex; the chosen vertex takes the role of v_* in the current branch. We then run the algorithm of Lemma 5.4.5 to compute the quantities $A_t^{\overline{c},\overline{w}}(f)$. At the root, we have that $L_{\hat{r}}^{live} = \emptyset$. The algorithm returns true if there is some branch and choice of $\overline{c} \leq \overline{b}$, $\overline{w} \in [0, 2n^2]$, such that $A_{\hat{r}}^{\overline{c},\overline{w}}(\emptyset) \neq 0$, otherwise the algorithm returns false.

The running time directly follows from Lemma 5.4.5. For the correctness, first note that at the root, we have $\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}(\emptyset) = \mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}$. Since all labels are dead at $\hat{r}$, we have for any $(X_L, X_R, F) \in \mathcal{A}_{\hat{r}}$ that $(X, (X_L, X_R)) \in \mathcal{C}(G)$, $v_* \in X_L$, $V \subseteq N_G[X]$ and hence $F = \emptyset$, where $X = X_L \cup X_R$. So, X is a dominating set of G that contains v_* and vice versa $\mathcal{A}_{\hat{r}}$ contains all consistent cuts of such dominating sets. The algorithm only returns true, if there are some $\overline{c} \in [0, \overline{b}]$, $\overline{w} \in [0, 2n^2]$ such that $\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}$ has odd cardinality. Defining $\mathcal{R} = \{X \subseteq V : V \subseteq N_G[X], v_* \in X\}$ and $\mathcal{R}^{\overline{c},\overline{w}} = \{X \in \mathcal{R} : |X| = \overline{c}, \mathbf{w}(X) = \overline{w}\}$, this implies that there exists a connected dominating set X of G with $\mathbf{c}(X) = \overline{c} \leq \overline{b}$ by Theorem 3.1.6, hence the algorithm does not return false positives.

For the error probability, suppose that the weight function $\mathbf{w}$ isolates an optimum connected dominating set X^* and that $\mathbf{c}(X^*) \leq \overline{b}$; by Lemma 3.1.4, this happens with probability $\geq 1/2$. In a branch with $v_* \in X^*$, we have that $X^* \in \mathcal{R}^{\overline{c},\overline{w}}$, where $\overline{c} = |X^*|$ and $\overline{w} = \mathbf{w}(X^*)$. By Lemma 3.1.1, the connected dominating set contributes an odd number to $|\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}|$ and all other contributing sets $Y \in \mathcal{R}^{\overline{c},\overline{w}}$ cannot be connected due to isolation and therefore contribute an even number to $|\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}|$. Therefore $A_{\hat{r}}^{\overline{c},\overline{w}}(\emptyset) \equiv_2 |\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}| = 1 \neq 0$ and hence the algorithm returns true with probability at least $1/2$ given a positive instance. □

5.5 Connected Deletion to q-Colorable Algorithm

We only consider Connected Deletion to q-Colorable instances where the costs are polynomially bounded in the input size. Furthermore, we assume without loss of generality that G is connected and contains at least two vertices.

Connected Deletion to q-Colorable generalizes the problems Connected Vertex Cover ($q = 1$) and Connected Odd Cycle Transversal ($q = 2$). We show how Connected Deletion to q-Colorable can be solved in time $\mathcal{O}^*((2^{q+2} - 2)^k)$ given a k-expression for G. To verify that the remaining graph $G - X$ is q-colorable, the objects (X, φ) considered by the dynamic programming algorithm consist of a deletion set X and a q-coloring φ of the remainder $G - X$. However, for a fixed X there might be multiple valid q-colorings φ. Therefore, to avoid unwanted cancellations due to multiple valid φ, the weight function used for the isolation lemma should also take φ into account. An additional side effect of considering the pairs (X, φ) is that our application of the cut-and-count-technique does not solve the modulo-2 counting variant of Connected Deletion to q-Colorable, whereas for many other problems, the essentially same cut-and-count-algorithm solves the decision version and the modulo-2 counting version.

Given a k-expression μ for $G = (V, E)$, we can assume, after polynomial-time preprocessing, that μ is a nice k-expression by Lemma 5.1.4. Fix q for this section. We want to apply the cut-and-count-technique to solve Connected Deletion to q-Colorable in time $\mathcal{O}^*((2^{q+2} - 2)^k)$. To do so, we first branch on the vertices of G, picking one vertex v_*, and in this branch only consider solutions containing v_*. We perform bottom-up dynamic programming along the augmented syntax tree $\hat{T}_\mu$. At every node $t \in V(\hat{T}_\mu)$, we consider the following family of partial solutions

$$\begin{aligned}\mathcal{A}_t = \{((X, \varphi), (X_L, X_R)) : (X, (X_L, X_R)) \in \mathcal{C}(G_t),\\ \varphi \text{ is a } q\text{-coloring of } G_t - X \text{ and}\\ v_* \in V_t \to v_* \in X_L\}.\end{aligned}$$

In other words, $\mathcal{A}_t$ contains all pairs of consistently cut deletion sets of G_t, with v_* on the left side of the cut if possible, together with a witnessing q-coloring of the remaining graph.

For the isolation lemma, cf. Lemma 3.1.4, we set the universe to $U = V \times (\{\mathbf{0}\} \cup [q])$, where $\mathbf{0}$ is a formal symbol representing the deletion of a vertex and the numbers in $[q]$ represent the colors. So, we sample a weight function $\mathbf{w}\colon U \to [N]$ with $N = 2(q + 1)n$ to guarantee an error probability of less than $1/2$ in the isolation lemma. The weight of a pair (X, φ) consisting of a deletion set X and q-coloring φ is given by $\mathbf{w}(X, \varphi) = \sum_{v \in X} \mathbf{w}(v, \mathbf{0}) + \sum_{v \in \mathtt{dom}(\varphi)} \mathbf{w}(v, \varphi(v))$. For every $t \in V(\hat{T}_\mu)$, $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(U)]$, we define $\mathcal{A}_t^{\overline{c},\overline{w}} = \{((X, \varphi), (X_L, X_R)) \in \mathcal{A}_t : \mathbf{c}(X) = \overline{c}, \mathbf{w}(X, \varphi) = \overline{w}\}$. Let $\hat{r}$ denote the root node of the augmented syntax tree $\hat{T}_\mu$. By Theorem 3.1.6, it follows that there exists a connected deletion set X of G such that $G - X$ is q-colorable and with $\mathbf{c}(X) \leq \overline{b}$ if there exist $\overline{c} \in [0, \overline{b}]$ and $\overline{w} \in [0, \mathbf{w}(U)]$ such that $\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}$ has odd cardinality.

We proceed by analyzing the possible label states. A single vertex v can take one of the states $\Omega = \{\mathbf{0}_L, \mathbf{0}_R\} \cup [q]$, where $\mathbf{0}_L$ and $\mathbf{0}_R$ represent deleted vertices on the left side or the right side of the cut respectively and the states in $[q]$ represent the colors of undeleted

vertices[3]. The state of a label V_t^ℓ, $\ell \in L_t$, is given by which vertex states appear in the label, thus the power set $\mathcal{P}(\Omega)$ captures all possible label states. However, we can exclude the state $\emptyset$, as we only save the state for nonempty labels. Moreover, we can exclude the state Ω at live labels, cf. Definition 5.1.7, as partial solutions assuming state Ω at a live label can never be extended to a solution of the whole graph. Therefore, we set $\mathbf{States} = \mathcal{P}(\Omega) \setminus \{\emptyset, \Omega\}$ and note that $|\mathbf{States}| = 2^{q+2} - 2$.

Given a node $t \in V(\hat{T}_\mu)$, a *t-signature* is a function $f\colon L_t^{live} \to \mathbf{States}$. For every node $t \in V(\hat{T}_\mu)$, $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(U)]$, and t-signature f, we define

$$\begin{aligned}\mathcal{A}_t^{\overline{c},\overline{w}}(f) = \{((X,\varphi),(X_L,X_R)) \in \mathcal{A}_t^{\overline{c},\overline{w}} :\ & f(\ell) \cap [q] = \varphi(V_t^\ell \setminus X) \text{ for all } \ell \in L_t^{live},\\ & \mathbf{0}_L \in f(\ell) \leftrightarrow X_L \cap V_t^\ell \neq \emptyset \text{ for all } \ell \in L_t^{live},\\ & \mathbf{0}_R \in f(\ell) \leftrightarrow X_R \cap V_t^\ell \neq \emptyset \text{ for all } \ell \in L_t^{live}\}.\end{aligned}$$

We proceed by giving the dynamic programming recurrences to compute the values $A_t^{\overline{c},\overline{w}}(f) = |\mathcal{A}_t^{\overline{c},\overline{w}}(f)| \mod 2$, for every $t \in V(\hat{T}_\mu)$, t-signature f, $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(U)]$ depending on the type of the considered node t. The recurrences for relabel nodes, dead nodes, and union nodes are essentially the same as for Connected Vertex Cover, hence we keep their explanation brief.

Introduce Node. If $t = \ell(v)$ for some $\ell \in [k]$, then $L_t^{live} = \{\ell\}$. We define the helper function $\mathrm{check}\colon V \times \mathbf{States} \times [0, \mathbf{c}(V)] \times [0, \mathbf{w}(U)] \to \{0,1\}$ defined by

$$\mathrm{check}(v, \mathbf{S}, \overline{c}, \overline{w}) = \begin{cases} 1, & \text{if } \mathbf{S} \in \{\{\mathbf{0}_L\}, \{\mathbf{0}_R\}\} \wedge \overline{c} = \mathbf{c}(v) \wedge \overline{w} = \mathbf{w}(v, \mathbf{0}),\\ 1, & \text{if } \mathbf{S} = \{i\} \wedge \overline{c} = 0 \wedge \overline{w} = \mathbf{w}(v,i) \text{ for some } i \in [q],\\ 0, & \text{else,}\end{cases}$$

which checks that vertex v gets assigned a singleton state $\mathbf{S} \in \mathbf{States}$, $|\mathbf{S}| = 1$, and appropriate cost $\overline{c}$ and weight $\overline{w}$. Adding edge case handling for v_*, we can formulate the recurrence for the introduce node as follows:

$$A_t^{\overline{c},\overline{w}}(f) = [v \neq v_* \vee f(\ell) = \{\mathbf{0}_L\}] \cdot \mathrm{check}(v, f(\ell), \overline{c}, \overline{w}).$$

Relabel Node. If $t = \rho_{i\to j}(G_{t'})$, where t' is the child of t, for some $i, j \in [k]$, then $i \in L_{t'}$, $j \in L_{t'}$, $L_t = L_{t'} \setminus \{i\}$ and either $i, j \in L_{t'}^{live}$ or $i, j \in L_{t'}^{dead}$.

- If labels i and j are live at t', then j is live at t and the recurrence is given by

$$A_t^{\overline{c},\overline{w}}(f) = \sum_{\substack{\mathbf{S}_1,\mathbf{S}_2 \in \mathbf{States}:\\ \mathbf{S}_1 \cup \mathbf{S}_2 = f(j)}} A_{t'}^{\overline{c},\overline{w}}(f[i \mapsto \mathbf{S}_1, j \mapsto \mathbf{S}_2]).$$

- If i and j are dead at t', then j is dead at t and we copy the previous table, i.e.,

$$A_t^{\overline{c},\overline{w}}(f) = A_{t'}^{\overline{c},\overline{w}}(f).$$

[3] These names diverge a little from our usual state naming scheme, but they simplify some notation regarding the handling of q-coloring. We emphasize that the deleted vertices determine the cost of a solution, even though their states are denoted $\mathbf{0}_L$ or $\mathbf{0}_R$.

Join Node. To check which solutions remain feasible after adding a join between two labels, we introduce a helper function $\mathrm{feas}\colon \mathbf{States} \times \mathbf{States} \to \{0,1\}$ defined by $\mathrm{feas}(\mathbf{S}_1, \mathbf{S}_2) = [\mathbf{0}_L \in \mathbf{S}_1 \to \mathbf{0}_R \notin \mathbf{S}_2][\mathbf{0}_R \in \mathbf{S}_1 \to \mathbf{0}_L \notin \mathbf{S}_2] \prod_{i\in[q]} [i \notin \mathbf{S}_1 \vee i \notin \mathbf{S}_2]$. There are two reasons for infeasibility: a join edge connects both sides of the cut, i.e., $\mathbf{1}_L$ appears on one side and $\mathbf{1}_R$ on the other, or a join edge connects two undeleted vertices of the same color. We have that $t = \eta_{i,j}(G_{t'})$ for some $i \neq j \in L_{t'}$ and where t' is the child of t. We must have $i, j \in L_{t'}^{live}$ and if the set of dead vertices changes a future dead node will handle it. Therefore, we only filter partial solutions that became infeasible:

$$A_t^{\overline{c},\overline{w}}(f) = \mathrm{feas}(f(i), f(j)) A_{t'}^{\overline{c},\overline{w}}(f).$$

Dead Node. We have that $t = \bot_\ell(G_{t'})$, where t' is the child of t, $\ell \notin L_t^{live}$, and $L_t^{live} = L_{t'}^{live} \setminus \{\ell\}$. The recurrence is given by

$$A_t^{\overline{c},\overline{w}}(f) = \sum_{\mathbf{S}\in\mathbf{States}} A_{t'}^{\overline{c},\overline{w}}(f[\ell \mapsto \mathbf{S}]).$$

Due to $\mathbf{States} \neq \mathcal{P}(\Omega)$, similar to the dead node recurrence of Connected Vertex Cover, this recurrence is again somewhat subtle as it relies on the fact that no partial solution can attain label state Ω at label class V_t^ℓ.

Union Node. We have that $t = G_{t_1} \oplus G_{t_2}$, where t_1 and t_2 are the children of t and we have $L_t^{live} = L_{t_1}^{live} \cup L_{t_2}^{live}$. Given a t-signature f, we consider the union-split $f_{t,1}$, $f_{t,2}$, $f_{t,12}$ of f at t, cf. Definition 5.1.11. For every label $\ell \in \tilde{L}_{t,12} = L_{t_1}^{live} \cap L_{t_2}^{live}$, we consider all states $\mathbf{S}_1, \mathbf{S}_2 \in \mathbf{States}$ such that $\mathbf{S}_1 \cup \mathbf{S}_2 = f(\ell)$, where $\mathbf{S}_i$ is the state of label ℓ at t_i. Furthermore, we have to distribute $\overline{c}$ and $\overline{w}$ among the partial solutions at t_1 and the partial solutions at t_2. Hence, we obtain the recurrence

$$A_t^{\overline{c},\overline{w}}(f) = \sum_{\substack{\overline{c}_1+\overline{c}_2=\overline{c}\\ \overline{w}_1+\overline{w}_2=\overline{w}}} \sum_{\substack{g_1,g_2\colon \tilde{L}_{t,12}\to\mathbf{States}\colon\\ g_1(\ell)\cup g_2(\ell)=f_{t,12}(\ell)\,\forall \ell\in\tilde{L}_{t,12}}} A_{t_1}^{\overline{c}_1,\overline{w}_1}(g_1 \cup f_{t,1}) A_{t_2}^{\overline{c}_2,\overline{w}_2}(g_2 \cup f_{t,2}),$$

where we interpret g_i and $f_{t,i}$ as sets in the expression $g_i \cup f_{t,i}$ for $i \in [2]$.

Since the set $\mathbf{States}$ is a closure difference, we can apply Theorem 5.2.6 after a few branching steps like for Connected Vertex Cover to compute the recurrence simultaneously for all $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$, and t-signatures f in time $\mathcal{O}^*(|\mathbf{States}|^{|L_t^{live}|})$.

Lemma 5.5.1. *Given a nice k-expression μ of $G = (V, E)$, there is an algorithm that computes the quantities $A_t^{\overline{c},\overline{w}}(f)$ for all nodes $t \in V(\hat{T}_\mu)$, all t-signatures f, and all $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(U)]$, in time $\mathcal{O}^*(|\mathbf{States}|^k) = \mathcal{O}^*((2^{q+2}-2)^k)$.*

Proof. The algorithm proceeds by bottom-up dynamic programming along the augmented syntax tree $\hat{T}_\mu$ of the nice k-expression μ and computes the quantities $A_t^{\overline{c},\overline{w}}(f)$ via the given recurrences. For an introduce node, relabel node, or join node, the recurrence for $A_t^{\overline{c},\overline{w}}(f)$ for fixed f, $\overline{c}$, and $\overline{w}$, can clearly be computed in polynomial time, since additions and multiplications in $\mathbb{Z}_2$ take constant time. For a union node t, we have argued how to compute the recurrences for all f, $\overline{c}$, and $\overline{w}$ simultaneously in time $\mathcal{O}^*(|\mathbf{States}|^{|L_t^{live}|})$. As μ

is a k-expression, we have $|L_t^{live}| \leq k$ for all $t \in V(\hat{T}_\mu)$ and in particular at most $|\mathbf{States}|^k$ t-signatures for any node $t \in V(T_\mu)$. Since $\mathbf{c}(V)$ is assumed to be polynomial in n and $\mathbf{w}(U) \leq |U|N \leq (q+1)nN = 2(q+1)^2 n = \mathcal{O}(n)$, the running time follows as μ consists of a polynomial number of operations.

It remains to prove the correctness of the recurrences. We omit the straightforward correctness proofs for introduce nodes, relabel nodes, join nodes, and union nodes. The correctness proof for the dead nodes follows the same idea as for Connected Vertex Cover. Consider a dead node $t = \perp_\ell(G_{t'})$, where t' is the child of t. We claim that $\mathcal{A}_t^{\overline{c},\overline{w}}(f) = \bigcup_{\mathbf{S} \in \mathbf{States}} \mathcal{A}_{t'}^{\overline{c},\overline{w}}(f[\ell \mapsto \mathbf{S}])$ for all t-signatures f, $\overline{c}$, and $\overline{w}$. Since the union on the right-hand side is disjoint, this implies the correctness of the recurrence. We continue with proving the claim. As the set on the left-hand side is defined by fewer constraints, the right-hand side is contained in the left-hand side.

For the other direction, suppose there exists some $((X,\varphi),(X_L,X_R)) \in \mathcal{A}_t^{\overline{c},\overline{w}}(f) \setminus \left(\bigcup_{\mathbf{S} \in \mathbf{States}} \mathcal{A}_{t'}^{\overline{c},\overline{w}}(f[\ell \mapsto \mathbf{S}])\right)$ and consider V_t^ℓ. We must have that $X_L \cap V_t^\ell \neq \emptyset$, $X_R \cap V_t^\ell \neq \emptyset$, and $\varphi(V_t^\ell \setminus X) = [q]$ as $V_t^\ell \neq \emptyset$ and all other cases are covered by $\mathbf{States}$. Let t^* be the join node that caused the dead node t to exist. The node t^* adds the final join incident to $V_t^\ell = V_{t^*}^\ell$, say between V_t^ℓ and some $V_t^i = V_{t^*}^i$, $i \neq \ell$. Since we have $V_t^i \neq \emptyset$, at least one of the statements $\varphi(V_t^i \setminus X) \neq \emptyset$, $X_L \cap V_t^i \neq \emptyset$, $X_R \cap V_t^i \neq \emptyset$ has to be satisfied. In the first case, φ cannot be a q-coloring of $G_t - X$ as $\varphi(V_t^i \setminus X) \subseteq [q]$ and hence some join edge has the same color at both endpoints. In the remaining two cases, there is a join edge crossing the cut (X_L, X_R). Therefore, we have derived a contradiction to $((X,\varphi),(X_L,X_R)) \in \mathcal{A}_t^{\overline{c},\overline{w}}(f)$ in all cases. Hence, $\mathcal{A}_t^{\overline{c},\overline{w}}(f) \subseteq \bigcup_{\mathbf{S} \in \mathbf{States}} \mathcal{A}_{t'}^{\overline{c},\overline{w}}(f[\ell \mapsto \mathbf{S}])$ which proves the claim. □

Theorem 5.5.2. *There is a randomized algorithm that given a nice k-expression μ for a graph $G = (V,E)$ can solve* Connected Deletion to q-Colorable *in time $\mathcal{O}^*(|\mathbf{States}|^k) = \mathcal{O}^*((2^{q+2}-2)^k)$. The algorithm does not return false positives and returns false negatives with probability at most $1/2$.*

Proof. We begin by sampling a weight function $\mathbf{w}\colon V \to [2N]$ uniformly at random. We branch over the vertices of G, picking one to take the role of v_* in the current branch. We then run the algorithm of Lemma 5.5.1 to compute the quantities $A_t^{\overline{c},\overline{w}}(f)$. Let $\hat{r}$ denote the root node of the expression μ. At the root, we have that $L_{\hat{r}}^{live} = \emptyset$. The algorithm returns true if in one of the branches there is some choice of $\overline{c} \leq \overline{b}$, $\overline{w} \in [0, \mathbf{w}(U)]$, such that $A_{\hat{r}}^{\overline{c},\overline{w}}(\emptyset) \neq 0$, otherwise the algorithm returns false.

The running time directly follows from Lemma 5.5.1. For the correctness, first note that at the root, we have $\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}(\emptyset) = \mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}$. The algorithm only returns true, if there are some $\overline{c} \in [0,\overline{b}]$, $\overline{w} \in [0,\mathbf{w}(U)]$ such that $\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}$ has odd cardinality. By Theorem 3.1.6, this implies the existence of an appropriate solution to Connected Deletion to q-Colorable, hence the algorithm does not return false positives.

For the error probability, suppose that the weight function $\mathbf{w}$ isolates the family $\mathcal{F}$ consisting of all (X,φ) such that X is a smallest connected deletion set such that $G - X$ is q-colorable and φ is a q-coloring of $G - X$. By Lemma 3.1.4, the isolation happens with probability greater than or equal to $|U|/N = 1/2$. Let (X^*,φ^*) denote the isolated element. Suppose that the considered problem instance has a positive answer, hence we have that $\mathbf{c}(X^*) \leq \overline{b}$. Furthermore, consider a branch with $v_* \in X^*$. Set $\overline{c} = \mathbf{c}(X^*) \leq \overline{b}$ and

$\overline{w} = \mathbf{w}(X^*, \varphi^*)$. By Lemma 3.1.1, the pair (X^*, φ^*) contributes once to $|\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}|$ and for every other contributing pair (X, φ) the graph $G[X]$ cannot be connected by isolation of (X^*, φ^*) and hence (X, φ) contributes an even number to $|\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}|$. Therefore $A_{\hat{r}}^{\overline{c},\overline{w}}(\emptyset) \equiv_2 |\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}| = 1 \neq 0$ and the algorithm returns true with probability at least $1/2$ given a positive instance. □

5.6 Steiner Tree Algorithm

We only consider STEINER TREE instances where the costs are polynomially bounded in the input size. Moreover, we assume without loss of generality that G is connected and contains at least two vertices.

By Lemma 5.1.4, we can assume that we are given a nice k-expression μ of G. We show how to solve STEINER TREE in time $\mathcal{O}^*(4^k)$ by dynamic programming along the syntax tree T_μ using the cut-and-count-technique. We use nice expressions to avoid some edge cases, but we do not distinguish live and dead labels for the STEINER TREE algorithm, hence it suffices to use the (unaugmented) syntax tree T_μ. We fix some terminal $v_* \in K$ and sample a weight function $\mathbf{w}\colon V \to [2|V|]$ for the isolation lemma, cf. Lemma 3.1.4. At every node $t \in V(T_\mu)$, we consider the following family of partial solutions

$$\mathcal{A}_t = \{(X, (X_L, X_R)) \in \mathcal{C}(G_t) : K \cap V_t \subseteq X \text{ and } v_* \in V_t \to v_* \in X_L\}.$$

So, $\mathcal{A}_t$ contains all consistently cut vertex sets of G_t containing the terminals in V_t and such that v_* is on the left side of the cut if possible. For every $t \in V(T_\mu)$, $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$, we define $\mathcal{A}_t^{\overline{c},\overline{w}} = \{(X, (X_L, X_R)) \in \mathcal{A}_t : \mathbf{c}(X) = \overline{c}, \mathbf{w}(X) = \overline{w}\}$. Let $\hat{r}$ denote the root node of the syntax tree T_μ. By Theorem 3.1.6, it follows that there exists a Steiner tree X of G with $\mathbf{c}(X) \leq \overline{b}$ if there exist $\overline{c} \in [0, \overline{b}]$ and $\overline{w} \in [0, \mathbf{w}(V)]$ such that $\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}$ has odd cardinality.

The vertex states of the treewidth-algorithm for STEINER TREE are $\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R$, meaning respectively that the vertex is not in the solution, on the left side of the solution, or on the right side of the solution. For STEINER TREE the basic state $\mathbf{0}$ does not impose any constraints on the neighboring vertices. Hence, it is sufficient to store for each label only which of the vertex states $\mathbf{1}_L$ and $\mathbf{1}_R$ occur. We do not exclude any further states and thus set $\Omega = \{\mathbf{1}_L, \mathbf{1}_R\}$ and $\mathbf{States} = \mathcal{P}(\Omega) = \{\emptyset, \{\mathbf{1}_L\}, \{\mathbf{1}_R\}, \{\mathbf{1}_L, \mathbf{1}_R\}\}$. Note that $\emptyset$ is a sensible label state for STEINER TREE, unless the considered label contains a terminal, as this represents that no vertex inside the considered label belongs to the partial solution.

Given a node $t \in V(T_\mu)$, a *t-signature* is a function $f\colon L_t \to \mathbf{States}$. For every node $t \in V(T_\mu)$, $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$, and t-signature f, we define

$$\mathcal{A}_t^{\overline{c},\overline{w}}(f) = \{(X, (X_L, X_R)) \in \mathcal{A}_t^{\overline{c},\overline{w}} : \mathbf{1}_L \in f(\ell) \leftrightarrow X_L \cap V_t^\ell \neq \emptyset \text{ for all } \ell \in L_t,$$
$$\mathbf{1}_R \in f(\ell) \leftrightarrow X_R \cap V_t^\ell \neq \emptyset \text{ for all } \ell \in L_t\}.$$

Instead of computing the sets $\mathcal{A}_t^{\overline{c},\overline{w}}(f)$ directly, we compute only the parity of their cardinality, i.e., $A_t^{\overline{c},\overline{w}}(f) = |\mathcal{A}_t^{\overline{c},\overline{w}}(f)| \mod 2$. We proceed to give recurrences for computing $A_t^{\overline{c},\overline{w}}(f)$, for every $t \in V(T_\mu)$, t-signature f, $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$ depending on the type of the considered node t.

Introduce Node. If $t = \ell(v)$ for some $\ell \in [k]$, then $A_t^{\overline{c},\overline{w}}(f)$ can be computed by

$$A_t^{\overline{c},\overline{w}}(f) = [v \neq v_* \vee f(\ell) = \{\mathbf{1}_L\}] \cdot [v \notin K \vee f(\ell) \in \{\{\mathbf{1}_L\}, \{\mathbf{1}_R\}\}]$$
$$\cdot\ [(f(\ell) = \emptyset \wedge \overline{c} = \overline{w} = 0) \vee (f(\ell) \in \{\{\mathbf{1}_L\}, \{\mathbf{1}_R\}\} \wedge \overline{c} = \mathbf{c}(v) \wedge \overline{w} = \mathbf{w}(v))],$$

since $\mathcal{A}_t^{\overline{c},\overline{w}}(f) \subseteq \{(\emptyset, (\emptyset, \emptyset)), (\{v\}, (\{v\}, \emptyset)), (\{v\}, (\emptyset, \{v\}))\}$ with some of these partial solutions removed if $v = v_*$ or $v \in K$.

Relabel Node. If $t = \rho_{i \to j}(G_{t'})$, where t' is the child of t, for some $i, j \in [k]$, then by niceness of μ it follows that $i \in L_{t'}$, $j \in L_{t'}$, $L_t = L_{t'} \setminus \{i\}$. Since $V_t^j = V_{t'}^i \cup V_{t'}^j$, we simply need to iterate over all possible state combinations yielding state $f(j)$ when combined:

$$A_t^{\overline{c},\overline{w}}(f) = \sum_{\substack{\mathbf{S}_1, \mathbf{S}_2 \in \mathbf{States}: \\ \mathbf{S}_1 \cup \mathbf{S}_2 = f(j)}} A_{t'}^{\overline{c},\overline{w}}(f[i \mapsto \mathbf{S}_1, j \mapsto \mathbf{S}_2]).$$

Join Node. To check whether two states can lead to a feasible solution after adding a join between their labels, we introduce a helper function $\mathrm{feas}\colon \mathbf{States} \times \mathbf{States} \to \{0,1\}$ defined by $\mathrm{feas}(\mathbf{S}_1, \mathbf{S}_2) = [\mathbf{1}_L \in \mathbf{S}_1 \to \mathbf{1}_R \notin \mathbf{S}_2][\mathbf{1}_R \in \mathbf{S}_1 \to \mathbf{1}_L \notin \mathbf{S}_2]$, or equivalently by the following table:

feas	$\emptyset$	$\{\mathbf{1}_L\}$	$\{\mathbf{1}_R\}$	$\{\mathbf{1}_L, \mathbf{1}_R\}$
$\emptyset$	1	1	1	1
$\{\mathbf{1}_L\}$	1	1	0	0
$\{\mathbf{1}_R\}$	1	0	1	0
$\{\mathbf{1}_L, \mathbf{1}_R\}$	1	0	0	0

The only reason for infeasibility is that a join edge connects both sides of the cut, i.e., $\mathbf{1}_L$ appears on one side and $\mathbf{1}_R$ on the other. If we have that $t = \eta_{i,j}(G_{t'})$ for some $i \neq j \in L_{t'}$ and where t' is the child of t, then we can simply filter out all partial solutions that became infeasible due to the new join:

$$A_t^{\overline{c},\overline{w}}(f) = \mathrm{feas}(f(i), f(j)) A_{t'}^{\overline{c},\overline{w}}(f).$$

Union Node. We have that $t = G_{t_1} \oplus G_{t_2}$, where t_1 and t_2 are the children of t and we have $L_t = L_{t_1} \cup L_{t_2}$. Given a t-signature f, we split f into three parts $\hat{f}_{t,1} = f\big|_{L_{t_1} \setminus L_{t_2}}$, $\hat{f}_{t,2} = f\big|_{L_{t_2} \setminus L_{t_1}}$, and $\hat{f}_{t,12} = f\big|_{L_{t_1} \cap L_{t_2}}$, similar to the union-split, cf. Definition 5.1.11. For every label $\ell \in L_{t_1} \cap L_{t_2}$, we must consider all states $\mathbf{S}_1, \mathbf{S}_2 \in \mathbf{States}$ such that $\mathbf{S}_1 \cup \mathbf{S}_2 = f(\ell)$, where $\mathbf{S}_i$ is the state of label ℓ at t_i. For the remaining labels, we essentially copy the previous state from the appropriate child of t. Furthermore, we need to distribute $\overline{c}$ and $\overline{w}$ among the partial solutions at t_1 and the partial solutions at t_2. Hence, we obtain the recurrence

$$A_t^{\overline{c},\overline{w}}(f) = \sum_{\substack{\overline{c}_1 + \overline{c}_2 = \overline{c} \\ \overline{w}_1 + \overline{w}_2 = \overline{w}}} \sum_{\substack{g_1, g_2 \colon L_{t_1} \cap L_{t_2} \to \mathbf{States}: \\ g_1(\ell) \cup g_2(\ell) = \hat{f}_{t,12}(\ell)\ \forall \ell \in L_{t_1} \cap L_{t_2}}} A_{t_1}^{\overline{c}_1,\overline{w}_1}(g_1 \cup \hat{f}_{t,1}) A_{t_2}^{\overline{c}_2,\overline{w}_2}(g_2 \cup \hat{f}_{t,2}),$$

where we interpret g_i and $\hat{f}_{t,i}$ as sets in the expression $g_i \cup \hat{f}_{t,i}$ for $i \in [2]$.

Similar to CONNECTED VERTEX COVER, this recurrence can be computed in time $\mathcal{O}^*(4^{|L_t|})$ for fixed $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$, and for all t-signatures f. We first branch on all possibilities for $(\overline{c}_1, \overline{c}_2, \overline{w}_1, \overline{w}_2)$; these are $n^{\mathcal{O}(1)}$ possibilities as $\mathbf{c}(V) \leq n^{\mathcal{O}(1)}$ by assumption and $\mathbf{w}(V) \leq 2n^2$. Fixing one of these possibilities, we further branch on $\hat{f}_{t,1}$ and $\hat{f}_{t,2}$, this leads to $4^{|L_{t_1} \setminus L_{t_2}| + |L_{t_2} \setminus L_{t_1}|}$ choices. Now, the quantities $A_{t_1}^{\overline{c}_1, \overline{w}_1}(g_1 \cup \hat{f}_{t,1})$ and $A_{t_2}^{\overline{c}_2, \overline{w}_2}(g_2 \cup \hat{f}_{t,2})$ can be viewed as functions of g_1 and g_2 respectively and the inner sum over these in the recurrence is their componentwise cover product over $\mathbf{States}$ evaluated at $\hat{f}_{t,12}$.

Since the set $\mathbf{States}$ is a power set and hence trivially a closure difference, we can apply Theorem 5.2.6 to compute this componentwise cover product in time $\mathcal{O}^*(4^{|L_{t_1} \cap L_{t_2}|})$ for all possible $\hat{f}_{t,12}$. Therefore, we need in total time $\mathcal{O}^*(4^{|L_t|})$ to compute $A_t^{\overline{c}, \overline{w}}(f)$ for all choices of $\overline{c}$, $\overline{w}$, and f.

Lemma 5.6.1. *Given a nice k-expression μ of $G = (V, E)$, there is an algorithm that computes the quantities $A_t^{\overline{c}, \overline{w}}(f)$ for all nodes $t \in V(T_\mu)$, all t-signatures f, and all $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, 2n^2]$, in time $\mathcal{O}^*(4^k)$.*

Proof. The algorithm proceeds by bottom-up dynamic programming along the syntax tree T_μ of the nice k-expression μ and computes the quantities $A_t^{\overline{c}, \overline{w}}(f)$ via the given recurrences. For an introduce node, relabel node, or join node, the recurrence for $A_t^{\overline{c}, \overline{w}}(f)$ for fixed f, $\overline{c}$, and $\overline{w}$, can clearly be computed in polynomial time, since additions and multiplications in $\mathbb{Z}_2$ take constant time. For a union node t, we have argued how to compute the recurrences for all f, $\overline{c}$, and $\overline{w}$ simultaneously in time $\mathcal{O}^*(4^{|L_t|})$. As μ is a k-expression, we have $|L_t| \leq k$ for all $t \in V(T_\mu)$ and in particular at most 6^k t-signatures for any node $t \in V(T_\mu)$. Therefore, the running time follows as μ consists of a polynomial number of operations. The proofs of correctness for the recurrences are straightforward and hence omitted. □

Theorem 5.6.2. *There is a randomized algorithm that given a nice k-expression μ for a graph $G = (V, E)$ can solve* STEINER TREE *in time $\mathcal{O}^*(4^k)$. The algorithm does not return false positives and returns false negatives with probability at most $1/2$.*

Proof. We sample a weight function $\mathbf{w} \colon V \to [2n]$ uniformly at random. We fix some terminal $v_* \in K$ and run the algorithm of Lemma 5.6.1 to compute the quantities $A_t^{\overline{c}, \overline{w}}(f)$. Let $\hat{r}$ denote the root node of the expression μ. The algorithm returns true if in one of the branches there is some choice of $\hat{r}$-signature f, costs $\overline{c} \leq \overline{b}$, and weights $\overline{w} \in [0, \mathbf{w}(V)]$, such that $A_{\hat{r}}^{\overline{c}, \overline{w}}(\emptyset) \neq 0$, otherwise the algorithm returns false.

The running time directly follows from Lemma 5.6.1. For the correctness, first note that the sets $\mathcal{A}_{\hat{r}}^{\overline{c}, \overline{w}}(f)$, ranging over f, partition $\mathcal{A}_{\hat{r}}^{\overline{c}, \overline{w}}$. The algorithm only returns true, if there are some $\hat{r}$-signature, $\overline{c} \in [0, \overline{b}]$, $\overline{w} \in [0, 2n^2]$ such that $\mathcal{A}_{\hat{r}}^{\overline{c}, \overline{w}}$ has odd cardinality. By Theorem 3.1.6, this implies that there is a Steiner tree X of G with $\mathbf{c}(X) = \overline{c} \leq \overline{b}$, hence the algorithm does not return false positives.

For the error probability, suppose that the weight function $\mathbf{w}$ isolates an optimum connected vertex cover X^* of G and that $\mathbf{c}(X^*) \leq \overline{b}$; by Lemma 3.1.4, the isolation happens with probability greater than or equal to $1/2$. Set $\overline{c} = \mathbf{c}(X^*) \leq \overline{b}$ and $\overline{w} = \mathbf{w}(X^*)$. By Lemma 3.1.1, the Steiner tree X^* contributes once to $|\mathcal{A}_{\hat{r}}^{\overline{c}, \overline{w}}(f)|$ for an appropriate $\hat{r}$-signature f. All other contributing sets cannot be connected due to isolation and hence contribute an even number to $|\mathcal{A}_{\hat{r}}^{\overline{c}, \overline{w}}(f)|$. Therefore $A_{\hat{r}}^{\overline{c}, \overline{w}}(f) \equiv_2 |\mathcal{A}_{\hat{r}}^{\overline{c}, \overline{w}}(f)| = 1 \neq 0$ and hence the algorithm returns true with probability at least $1/2$ given a positive instance. □

6 Algorithms Parameterized By Modular-Treewidth

6.1 Dynamic Programming for Modular-Treewidth

Consider a graph $G = (V, E)$ of modular-treewidth k in this subsection. We are given a tree decomposition of width k for every prime quotient graph $G^q_M \in \mathcal{H}_p(G)$. This allows for a dynamic programming scheme that works roughly as follows. We proceed bottom-up along the modular decomposition tree solving a subproblem for each node corresponding to an induced subgraph $G[M]$ for some $M \in \mathcal{M}_{\text{tree}}(G)$; this is also called the *outer dynamic program*. At internal nodes of the modular decomposition tree, we distinguish based on the three types given by Theorem 2.4.7 and solve the subproblem based on the results of the child nodes. If the node is a parallel or series node, then there often is a simple polynomial-time algorithm for this computation. On the other hand, if the node is a prime node, then we can use the given tree decomposition of the associated quotient graph and perform dynamic programming along this tree decomposition to solve the subproblem for the considered node of the modular decomposition tree; this is called the *inner dynamic program*. In several cases, the modular structure has such strong implications on the problem structure that we do not need to perform this thorough dynamic programming scheme. Instead, it can suffice to just consider the topmost quotient graph of the modular decomposition and just perform a single dynamic program along its tree decomposition. We now go into some more detail and give the basic definitions needed for the dynamic programming algorithms with respect to modular-treewidth.

Canonical Projection. When solving a subproblem at a nontrivial module $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$, we need to understand how subsets $X \subseteq M^\uparrow$ interact with the quotient graph $G^q_{M^\uparrow}$. We formalize this as follows. To each quotient graph $G^q_{M^\uparrow} = G[M^\uparrow]/\Pi_{mod}(G[M^\uparrow])$, $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$, appearing in the modular decomposition, we associate a *canonical projection* $\pi_{M^\uparrow} \colon M^\uparrow \to V(G^q_{M^\uparrow})$ with $\pi_{M^\uparrow}(v) = v^q_M$ whenever $v \in M \in \Pi_{mod}(G[M^\uparrow])$. Given a subset $X \subseteq M^\uparrow$, the set $\pi_{M^\uparrow}(X) = \{v^q_M : M \in \texttt{children}(M^\uparrow), X \cap M \neq \emptyset\}$ describes the interaction of X with the quotient graph $G^q_{M^\uparrow}$. For a fixed module $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$, we will often use the shorthand $X^q = \pi_{M^\uparrow}(X)$.

Associated Subgraphs for Modular-Treewidth. To define the set of considered partial solutions in the inner dynamic program along the tree decomposition, we must explain which subgraph corresponds to each node of the tree decomposition. Given a very nice tree decomposition $(\mathcal{T}^q_{M^\uparrow}, (B^q_t)_{t \in V(\mathcal{T}^q_{M^\uparrow})})$ of the quotient graph $G^q_{M^\uparrow}$, we associate with every node $t \in V(\mathcal{T}^q_{M^\uparrow})$ a subgraph $G^q_t = (V^q_t, E^q_t)$ of $G^q_{M^\uparrow}$ as follows:

- V_t^q contains all $v_M^q \in V(G_{M^\uparrow}^q)$ such that there is a descendant t' of t in $\mathcal{T}_{M^\uparrow}^q$ with $v_M^q \in B_{t'}^q$,
- E_t^q contains all $\{v_{M_1}^q, v_{M_2}^q\} \in E(G_{M^\uparrow}^q)$ that were introduced by a descendant of t in $\mathcal{T}_{M^\uparrow}^q$.

At the root node $\hat{r}$ of $\mathcal{T}_{M^\uparrow}^q$, we have that $G_{\hat{r}}^q = G_{M^\uparrow}^q$. By expanding the vertices of the quotient graph back into modules, we can lift the bags B_t^q to vertex subsets of $M^\uparrow$ and the subgraphs G_t^q to subgraphs of $G[M^\uparrow]$ as follows. We define $B_t = \pi_{M^\uparrow}^{-1}(B_t^q) = \bigcup_{v_M^q \in B_t^q} M$ and $V_t = \pi_{M^\uparrow}^{-1}(V_t^q) = \bigcup_{v_M^q \in V_t^q} M$. We also transfer the edge set E_t^q by using all edges inside modules M with $v_M^q \in V_t^q$ and expanding each edge $\{v_{M_1}^q, v_{M_2}^q\} \in E_t^q$ into the corresponding join between M_1 and M_2, i.e., we define

$$E_t = \bigcup_{v_M^q \in V_t^q} E(G[M]) \cup \bigcup_{\{v_{M_1}^q, v_{M_2}^q\} \in E_t^q} \{\{u_1, u_2\} : u_1 \in M_1 \wedge u_2 \in M_2\},$$

allowing us to define the graph $G_t = (V_t, E_t)$ associated to any node $t \in V(\mathcal{T}_{M^\uparrow}^q)$.

Signatures. Given a tree decomposition $(\mathcal{T}_{M^\uparrow}^q, (B_t^q)_{t \in V(\mathcal{T}_{M^\uparrow}^q)})$ of the quotient graph $G_{M^\uparrow}^q$ for some $M \in \mathcal{M}_{\text{tree}}^*(G)$, the inner dynamic program distinguishes partial solutions at node $t \in V(\mathcal{T}^q)$ based on their state at the current bag. We encode this state by *signatures* of the form $f \colon B_t^q \to \mathbf{States}$, for some finite set $\mathbf{States}$ of module states. Note that B_t^q consists of vertices from the quotient graph, but the intended meaning of $f(v_M^q) = \mathbf{s}$ for some $\mathbf{s} \in \mathbf{States}$ is that the corresponding module $M \in \texttt{children}(M^\uparrow)$ has state $\mathbf{s}$.

6.1.1 Cut and Count for Modular-Treewidth

To apply the cut-and-count-technique for modular-treewidth, we first study how connectivity interacts with the modular structure. Typically, we consider vertex sets X contained in some module $M^\uparrow \in \mathcal{M}_{\text{tree}}^*(G)$ that intersect at least two child modules of $M^\uparrow$, i.e., $|\pi_{M^\uparrow}(X)| \geq 2$. When $|\pi_{M^\uparrow}(X)| = 1$, we can recurse in the modular decomposition tree until at least two child modules are intersected or we arrive at an easily solvable special case. The following exchange argument shows that the connectivity of $G[X]$ is not affected by the precise intersection $X \cap M$, $M \in \texttt{children}(M^\uparrow)$, but only whether $X \cap M$ is empty or not.

Lemma 6.1.1. *Let $M^\uparrow \in \mathcal{M}_{\text{tree}}^*(G)$ and $X \subseteq M^\uparrow$ be a subset with $|\pi_{M^\uparrow}(X)| \geq 2$ and such that $G[X]$ is connected. For any module $M \in \texttt{children}(M^\uparrow)$ with $X \cap M \neq \emptyset$ and $\emptyset \neq Y \subseteq M$, the graph $G[(X \setminus M) \cup Y]$ is connected.*

Proof. Since $G[X]$ is connected and intersects at least two modules, there has to be a module $M' \in \texttt{children}(M^\uparrow)$ adjacent to M such that $X \cap M' \neq \emptyset$. The edges between Y and $X \cap M'$ induce a biclique and hence all incident vertices must be connected to each other. Fix a vertex $u \in X \cap M$ and consider any $w \in X \setminus M$, then $G[X]$ contains an u,w-path P such that the vertex v after u on P is in $X \setminus M$. For any $y \in Y$, we obtain an y,w-path P_y in $G[(X \setminus M) \cup Y]$ by replacing u with y in P. Finally, consider two vertices $u, w \in X \setminus M$, then there is an u,w-path P in $G[X]$. If P does not intersect M, then P is also a path in $G[(X \setminus M) \cup Y]$. Otherwise, we can assume that P contains exactly one vertex v of M and simply replace v with some $y \in Y$ to obtain a u,w-path P' in $G[(X \setminus M) \cup Y]$. Hence, $G[(X \setminus M) \cup Y]$ is connected as claimed. □

Building upon Lemma 6.1.1 allows us to reduce checking the connectivity of $G[X]$ to the quotient graph at $M^\uparrow$, as $G^q_{M^\uparrow}$ is isomorphic to the induced subgraph of G obtained by picking one vertex from each child module of $M^\uparrow$.

Lemma 6.1.2. *Let $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$ and $X \subseteq M^\uparrow$ with $|\pi_{M^\uparrow}(X)| \geq 2$, i.e., X intersects at least two modules in $\texttt{children}(M^\uparrow)$. It holds that $G[X]$ is connected if and only if $G^q_{M^\uparrow}[\pi_{M^\uparrow}(X)]$ is connected.*

Proof. For every module $M \in \texttt{children}(M^\uparrow)$ with $X \cap M \neq \emptyset$, pick a vertex $v_M \in X \cap M$ and define $X' = \{v_M : X \cap M \neq \emptyset, M \in \texttt{children}(M^\uparrow)\} \subseteq X$. Note that $G[X']$ is isomorphic to $G^q_{M^\uparrow}[\pi_{M^\uparrow}(X)]$. Hence, we are done if we can show that $G[X]$ is connected if and only if $G[X']$ is connected. If $G[X]$ is connected, then so is $G[X']$ by repeatedly applying Lemma 6.1.1.

For the converse, suppose that $G[X']$ is connected. We argue that every $v \in X \setminus X'$ is adjacent to some $w \in X'$ and then it follows that $G[X]$ is connected as well. There is some $M \in \texttt{children}(M^\uparrow)$ with $v \in M$ and $v_M \in X'$ by definition of X'. Since $|X'| \geq 2$ and $G[X']$ is connected, there is a neighbor $w \in X'$ of v_M in $G[X']$ and $w = v_{M'}$ for some $M' \in \texttt{children}(M^\uparrow) \setminus \{M\}$. The vertex w has to be a neighbor of v because M is a module and $w \notin M$. □

Lemma 6.1.2 tells us that *heterogeneous* cuts, i.e., $(X, (X_L, X_R)) \in \mathcal{C}(G)$ with $X_L \cap M \neq \emptyset$ and $X_R \cap M \neq \emptyset$ for some module $M \in \Pi_{mod}(G)$, do not need to be considered, as checking connectivity can be reduced to a set that contains at most one vertex per module.

Definition 6.1.3. Let $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$. We say that a cut (X_L, X_R), with $X_L \cup X_R \subseteq M^\uparrow$, is *$M^\uparrow$-homogeneous* if $X_L \cap M = \emptyset$ or $X_R \cap M = \emptyset$ for every $M \in \texttt{children}(M^\uparrow)$. We may just say that (X_L, X_R) is *homogeneous* when $M^\uparrow$ is clear from the context. We define for every subgraph G' of G the set $\mathcal{C}^{hom}_{M^\uparrow}(G') = \{(X, (X_L, X_R)) \in \mathcal{C}(G') : (X_L, X_R)$ is $M^\uparrow$-homogeneous$\}$.

Combining Lemma 3.1.1 with Lemma 6.1.2, the connectivity of $G[X]$ can be determined by counting $M^\uparrow$-homogeneous consistent cuts of $G[X]$ modulo 4.

Lemma 6.1.4. *Let $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$ and $X \subseteq M^\uparrow$ with $|\pi_{M^\uparrow}(X)| \geq 2$. It holds that $|\{(X_L, X_R) : (X, (X_L, X_R)) \in \mathcal{C}^{hom}_{M^\uparrow}(G)\}| = 2^{\texttt{cc}(G^q_{M^\uparrow}[\pi_{M^\uparrow}(X)])}$ and $G[X]$ is connected if and only if $|\{(X_L, X_R) : (X, (X_L, X_R)) \in \mathcal{C}^{hom}_{M^\uparrow}(G)\}| \neq 0 \mod 4$.*

Proof. Fix $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$ and $X \subseteq M^\uparrow$ with $|\pi_{M^\uparrow}(X)| \geq 2$. For any set $S \subseteq M^\uparrow$, we write $S^q = \pi_{M^\uparrow}(S)$ in this proof. We will argue that the map $(X_L, X_R) \mapsto (X^q_L, X^q_R)$ is a bijection between $\{(X_L, X_R) : (X, (X_L, X_R)) \in \mathcal{C}^{hom}_{M^\uparrow}(G)\}$ and $\{(Y_L, Y_R) : (X^q, (Y_L, Y_R)) \in \mathcal{C}(G^q_{M^\uparrow})\}$. First of all, notice that (X^q_L, X^q_R) is a cut of $G^q_{M^\uparrow}[X^q]$ because (X_L, X_R) is homogeneous. Furthermore, (X^q_L, X^q_R) is a consistent cut, since any edge $\{v^q_{M_1}, v^q_{M_2}\}$ crossing (X^q_L, X^q_R) would give rise to an edge $\{u_1, u_2\}$, $u_i \in M_i$, $i \in [2]$, crossing (X_L, X_R) which would therefore contradict the assumption that (X_L, X_R) is a consistent cut.

For injectivity, consider $(X, (X_L, X_R))$, $(X, (Z_L, Z_R)) \in \mathcal{C}^{hom}_{M^\uparrow}(G)$ with $(X^q_L, X^q_R) = (Z^q_L, Z^q_R)$. Since they are homogeneous cuts, we can compute

$$X_L = \bigcup_{v^q_M \in X^q_L} X \cap M = \bigcup_{v^q_M \in Z^q_L} X \cap M = Z_L$$

and similarly for $X_R = Z_R$. For surjectivity, note that (Y_L, Y_R) with $(X^q, (Y_L, Y_R)) \in \mathcal{C}(G^q_{M^\uparrow})$ is hit by the homogeneous cut $(X, (\bigcup_{v^q_M \in Y_L} X \cap M, \bigcup_{v^q_M \in Y_R} X \cap M))$.

Finally, we can apply Lemma 3.1.1 to $X^q \subseteq V(G^q_{M^\uparrow})$ to obtain, via the bijection, that $|\{(X, (X_L, X_R)) \in \mathcal{C}^{hom}_{M^\uparrow}(G)\}| = 2^{\mathsf{cc}(G^q_{M^\uparrow}[X^q])}$. Hence, $G^q_{M^\uparrow}[X^q]$ is connected if and only if $|\{(X, (X_L, X_R)) \in \mathcal{C}^{hom}_{M^\uparrow}(G)\}| \neq 0 \mod 4$. The statement then follows by Lemma 6.1.2. □

6.2 Independent Set Algorithm

We consider INDEPENDENT SET[modular-treewidth] or, dually, VERTEX COVER[modular-treewidth] and design an $\mathcal{O}^*(2^k)$-time algorithm. This algorithm serves as an easy example for dynamic programming along the tree decomposition of the quotient graphs and the algorithms for CONNECTED VERTEX COVER[modular-treewidth] and FEEDBACK VERTEX SET[modular-treewidth] will use this algorithm as a subroutine. Let $G = (V, E)$ be a graph with a cost function $\mathbf{c}\colon V \to \mathbb{N} \setminus \{0\}$. We show how to compute for every $M \in \mathcal{M}_{\text{tree}}(G)$ an independent set $X_M \subseteq M$ of $G[M]$ of maximum cost[1] in time $\mathcal{O}^*(2^k)$ given a tree decomposition of width at most k for every prime quotient graph $G^q_M \in \mathcal{H}_p(G)$ of G. We remark that for the problem variant without costs, this result already follows by the algorithm for INDEPENDENT SET[multi-clique-width] of Fürer [79] and Lemma 2.4.20. We first deal with the edge cases of the outer dynamic program, i.e., the leaf nodes of the modular decomposition tree, parallel nodes, and series nodes. Following that, we consider prime nodes and present the inner dynamic program along the tree decomposition of the corresponding quotient graph. Afterwards, we can bring all cases together and the algorithm follows.

Leaf Nodes. The leaf nodes of the modular decomposition tree are the base cases of the outer dynamic program. Here, we have some $M \in \mathcal{M}_{\text{tree}}(G)$ with $|M| = 1$ and therefore $M = \{v\}$ for some $v \in V$. Clearly, $X_{\{v\}} = \{v\}$ is an independent set of maximum cost of $G[\{v\}]$ in this case, which already concludes the base case.

Problem Structure Relative to Modules. Having solved the base case, we can proceed with computing X_M for internal nodes $M \in \mathcal{M}^*_{\text{tree}}(G) = \mathcal{M}_{\text{tree}}(G) \setminus \{\{v\} : v \in V\}$. Fixing some $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$, we can assume that we have already solved the subproblems corresponding to the child nodes $M \in \texttt{children}(M^\uparrow) = \Pi_{mod}(G[M^\uparrow])$ of $M^\uparrow$, i.e., X_M is computed for all $M \in \texttt{children}(M^\uparrow)$. Our task is to compute $X_{M^\uparrow}$ based on this data. The next lemma shows how an independent set interacts with the child nodes $M \in \texttt{children}(M^\uparrow)$, which confirms that the X_M, $M \in \texttt{children}(M^\uparrow)$, are indeed useful to compute $X_{M^\uparrow}$. For the rest of the algorithm description, we assume without loss of generality that $M^\uparrow = V$ is the root node and hence $\texttt{children}(M^\uparrow) = \texttt{children}(V) = \Pi_{mod}(G)$ to simplify the notation. We denote the quotient graph at the root node by $G^q := G^q_V = G/\Pi_{mod}(G)$.

[1] Even though we are considering a maximization problem, we still refer to the weights of the objective as *costs* for the sake of consistency.

Lemma 6.2.1. *If X is an independent set of G, then for every module $M \in \Pi_{mod}(G)$ either $X \cap M = \emptyset$ or $X \cap M$ is a nonempty independent set of $G[M]$. Furthermore, $X^q := \pi_V(X) = \{v^q_M : X \cap M \neq \emptyset\}$ is an independent set of G^q.*

Proof. If $G[X \cap M]$ contains an edge, then so does $G[X]$, hence the first part is trivially true. If $G^q[X^q]$ contains an edge $\{v^q_M, v^q_{M'}\}$, then M and M' are adjacent modules and $X \cap M \neq \emptyset$ and $X \cap M' \neq \emptyset$, so $G[X]$ cannot be an independent set. □

Parallel and Series Nodes. If V is a parallel or series node in the modular decomposition tree, i.e., G^q is an independent set or clique respectively, then we give special polynomial time algorithms to compute X_V without relying on a tree decomposition. If G^q is a parallel node, then we simply set $X_V = \bigcup_{M \in \Pi_{mod}(G)} X_M$. If G^q is a series node, then any independent set may intersect at most one module $M \in \Pi_{mod}(G)$, else the set would immediately induce an edge. Thus, we set in this case $X_V = \arg\max_{X_M} \mathbf{c}(X_M)$, where the maximum ranges over all X_M with $M \in \Pi_{mod}(G)$. Since each X_M is an independent set of maximum cost of $G[M]$, it is easy to see by using Lemma 6.2.1 that X_V must be an independent set of G of maximum cost in both cases.

Prime Nodes. If V is a prime node, i.e., the quotient graph $G^q = (V^q, E^q)$ is prime, then we are given a tree decomposition $(\mathcal{T}^q, (B^q_t)_{t \in V(\mathcal{T}^q)})$ of G^q of width at most k, which we can assume to be *very nice* by Lemma 2.4.2. We perform dynamic programming along this tree decomposition, yielding the inner dynamic program. By Lemma 6.2.1, it is natural that every module in the currently considered bag has two possible states; it can be empty (state **0**), or nonempty (state **1**) and in the latter case we take an independent set of maximum cost inside. Given that we have already computed the maximum independent sets X_M for each $M \in \Pi_{mod}(G)$, we define the partial solutions of the dynamic programming as follows.

For each node $t \in V(\mathcal{T}^q)$ of the tree decomposition, we define the family of *partial solutions at t*, denoted $\mathcal{A}_t$, as the family consisting of all $X \subseteq V_t = \pi_V^{-1}(V^q_t)$ such that the following properties hold for all $M \in \Pi_{mod}(G)$:

- $X \cap M \in \{\emptyset, X_M\}$,
- if $X \cap M \neq \emptyset$, then $X \cap M' = \emptyset$ for all $\{v^q_M, v^q_{M'}\} \in E(G^q_t)$.

We emphasize that while the tree decomposition is of the quotient graph G^q, the defined partial solutions live inside the original graph G. Using the already computed data X_M, $M \in \Pi_{mod}(G)$, and Lemma 6.2.1, the quotient graph structure is sufficient for constructing solutions of the whole graph G. Note that at the root node $\hat{r}$ of the tree decomposition $\mathcal{T}^q$, the family $\mathcal{A}_{\hat{r}}$ consists of all independent sets X of G with $X \cap M \in \{\emptyset, X_M\}$ for all $M \in \Pi_{mod}(G)$, which must in particular contain at least one independent set of maximum cost of G by Lemma 6.2.1 and the definition of X_M, $M \in \Pi_{mod}(G)$.

To set up the dynamic programming recurrences, we need to partition the partial solutions according to their interaction with the current bag. Given a *t-signature* $f \colon B^q_t \to \mathbf{States} := \{\mathbf{0}, \mathbf{1}\}$, the subfamily $\mathcal{A}_t(f) \subseteq \mathcal{A}_t$ of *partial solutions at t compatible with f* consists of all $X \in \mathcal{A}_t$ such that the following properties hold for all $v^q_M \in B^q_t$:

- $f(v^q_M) = \mathbf{0}$ implies that $X \cap M = \emptyset$,
- $f(v^q_M) = \mathbf{1}$ implies that $X \cap M = X_M$.

For each $t \in V(\mathcal{T}^q)$ and t-signature $f\colon B_t^q \to \mathbf{States}$, we compute the quantity $A_t(f) := \max_{X \in \mathcal{A}_t(f)} \mathbf{c}(X)$ by dynamic programming along the tree decomposition. The quantity $A_{\hat{r}}(\emptyset)$ at the root node $\hat{r}$ of the tree decomposition is the maximum cost of an independent set of G. By standard dynamic programming techniques, which we do not expand on here, we can also compute an explicit independent set X_V of G of maximum cost. The dynamic programming recurrences depend on the bag type of node t and are as follows.

Leaf Bag. Leaf bags are the base case of the inner dynamic program. We have $B_t = B_t^q = \emptyset$ and t is a leaf node of the tree decomposition, i.e., t has no children. Here, we simply have $\mathcal{A}_t = \emptyset$ and hence $A_t(\emptyset) = 0$.

Introduce Vertex Bag. We have that $B_t^q = B_s^q \cup \{v_M^q\}$ and $v_M^q \notin B_s^q$, where s is the only child node of t. We extend every s-signature by one of the two possible states for v_M^q and update the cost if necessary. Note that no edges incident to v_M^q are introduced yet. Hence, the recurrence is given by

$$A_t(f[v_M^q \mapsto \mathbf{0}]) = A_s(f),$$
$$A_t(f[v_M^q \mapsto \mathbf{1}]) = A_s(f) + \mathbf{c}(X_M),$$

where f is an s-signature.

Introduce Edge Bag. Let the introduced edge be denoted by $\{v_M^q, v_{M'}^q\}$. We have that $\{v_M^q, v_{M'}^q\} \subseteq B_t^q = B_s^q$, where s is the only child node of t. The recurrence only needs to filter all partial solutions X that intersect both M and M', since these cannot be independent sets. Hence, the recurrence is given by

$$A_t(f) = [f(v_M^q) = \mathbf{0} \vee f(v_{M'}^q) = \mathbf{0}]A_s(f),$$

where f is a t-signature.

Forget Vertex Bag. We have that $B_t^q = B_s^q \setminus \{v_M^q\}$ and $v_M^q \in B_s^q$, where s is the only child node of t. We simply try both states for the forgotten module M and take the maximum, so the recurrence is given by

$$A_t(f) = \max(A_s(f[v_M^q \mapsto \mathbf{0}]), A_s(f[v_M^q \mapsto \mathbf{1}])),$$

where f is a t-signature.

Join Bag. We have that $B_t^q = B_{s_1}^q = B_{s_2}^q$, where s_1 and s_2 are the two children of t. For each t-signature f, we can simply combine a best partial solution compatible with f at s_1 with one at s_2, but we do have to account for overcounting in the cost. We have that

$V^q_{s_1} \cap V^q_{s_2} = B^q_t$, so these partial solutions can only overlap in the current bag. Hence, the recurrence is given by

$$A_t(f) = A_{s_1}(f) + A_{s_2}(f) - \sum_{v^q_M \in f^{-1}(\mathbf{1})} \mathbf{c}(X_M),$$

where f is a t-signature.

Theorem 6.2.2. *There exists an algorithm that given a tree decomposition of width at most k for every prime quotient graph $G^q_M \in \mathcal{H}_p(G)$, solves* Independent Set *in time $\mathcal{O}^*(2^k \log N_c)$, where N_c is the maximum value attained by the cost function* $\mathbf{c}$.

Proof. We first compute the modular decomposition tree in time $\mathcal{O}(n+m)$ using the algorithm of Tedder et al. [169]. We perform the outer dynamic programming algorithm, solving the subproblems for leaf nodes, parallel nodes, and series nodes as described. For a prime node $M^\uparrow$, we use the given (very nice) tree decomposition of $G^q_{M^\uparrow} = G[M^\uparrow]/\Pi_{mod}(G[M^\uparrow])$ of width at most k and perform the inner dynamic programming algorithm as described, which computes a set $X_{M^\uparrow}$ by standard augmentation of the dynamic program. The algorithm returns true if $\mathbf{c}(X_V) \geq \bar{b}$ and false otherwise.

The correctness proofs of the recurrences for the inner dynamic program are straightforward and hence omitted. We have already argued the correctness of the algorithms for leaf, parallel, and series nodes, therefore it follows that X_V is an independent set of maximum cost of G and that the algorithm is correct.

It remains to consider the running time of the algorithm. Since $\mathbf{c}(V) \leq n \cdot N_c$ and we only add, subtract, or compare numbers, all number operations can be performed in time $\mathcal{O}(\log n + \log N_c)$. The modular decomposition tree contains $\mathcal{O}(n)$ nodes, as there are n leaves and every internal node has at least two child nodes. By standard arguments, we can also assume that each very nice tree decomposition has at most $\mathcal{O}(kn)$ nodes. Since every bag of the considered tree decompositions contains at most $k+1$ vertices, there are at most 2^{k+1} signatures at each node. Finally, the running time follows because every recurrence of the inner dynamic program only uses a polynomial number of operations. □

Lexicographic Maximum Independent Set. When using this algorithm as a subroutine, we want to find an independent set X that lexicographically maximizes $(\tilde{\mathbf{c}}(X), \tilde{\mathbf{w}}(X))$, where $\tilde{\mathbf{c}} \colon V \to [1, N_c]$ and $\tilde{\mathbf{w}} \colon V \to [1, N_w]$ are some given cost and weight function with maximum value N_c and N_w respectively. Setting $\mathbf{c}(v) = (|V|+1)N_w\tilde{\mathbf{c}}(v) + \tilde{\mathbf{w}}(v)$ for all $v \in V$, we can simulate this setting with a single cost function $\mathbf{c}$ and recover $\tilde{\mathbf{w}}(X) = \mathbf{c}(X) \bmod (|V|+1)N_w$ and $\tilde{\mathbf{c}}(X) = (\mathbf{c}(X) - \tilde{\mathbf{w}}(X))/((|V|+1)N_w)$. Alternatively, we may augment the dynamic programming to remember which arguments in the recurrences lead to the maximum to construct the independent set X and simply compute the values $\tilde{\mathbf{c}}(X)$ and $\tilde{\mathbf{w}}(X)$ directly.

Theorem 6.2.3. *Let $G = (V, E)$ be a graph, $\tilde{\mathbf{c}} \colon V \to [1, N_c]$ be a cost function, and $\tilde{\mathbf{w}} \colon V \to [1, N_w]$ be a weight function. If $N_c, N_w \leq |V|^{\mathcal{O}(1)}$, then there exists an algorithm that given a tree decomposition of width k for every prime quotient graph $G^q_M \in \mathcal{H}_p(G)$, computes an independent set X of G lexicographically maximizing $(\tilde{\mathbf{c}}(X), \tilde{\mathbf{w}}(X))$ in time $\mathcal{O}^*(2^k)$.*

Proof. We first transform $\tilde{\mathbf{c}}$ and $\tilde{\mathbf{w}}$ into a single cost function $\mathbf{c}$ as described and then run the algorithm of Theorem 6.2.2, obtaining the independent set X as described. The running time follows, since by the polynomial bounds on N_c and N_w, also the maximum value of $\mathbf{c}$ must be polynomially bounded. □

6.3 Steiner Tree Reduction

When solving the Steiner Tree problem, we assume that G is a connected graph, otherwise the answer is trivially no if the terminals are distributed across several connected components, or we can just look at the connected component containing all terminals. We also assume that $G[K]$ is not connected, as otherwise $X = K$ is trivially an optimal solution. Furthermore, we assume that the costs $\mathbf{c}(v)$, $v \in V$, are at most polynomial in $|V|$.

For Steiner Tree, it is sufficient to consider the topmost quotient graph $G^q := G^q_V = G/\Pi_{mod}(G)$, unless there is a single module $M \in \Pi_{mod}(G) = \texttt{children}(V)$ containing all terminals. In this edge case, we find a solution of size $|K|+1$, by taking a vertex in a module adjacent to M, or we consider the graph $G[M]$, allowing us to recurse into the module M.

We first consider the case that all terminals are contained in a single module $M \in \Pi_{mod}(G)$. The next lemma shows that we can either find a solution of size $|K|+1$, which can be computed in polynomial time, or it suffices to consider the graph $G[M]$.

Lemma 6.3.1. *If there is a module $M \in \Pi_{mod}(G)$ of G such that $K \subseteq M$, then there is an optimum Steiner tree X satisfying $X \subseteq M$, or there is an optimum Steiner tree X satisfying $|X| = |K| + 1$.*

Proof. Consider a Steiner tree X such that $X \not\subseteq M$, then X has to contain at least one vertex v inside a module $M' \in \Pi_{mod}(G)$ adjacent to M. We claim that $X' = K \cup \{v\}$ is a Steiner tree with $\mathbf{c}(X') \leq \mathbf{c}(X)$. Clearly, $X' \subseteq X$, and since the costs are positive we have that $\mathbf{c}(X') \leq \mathbf{c}(X)$. Since $K \subseteq M$, the vertex v is adjacent to all terminals K and $G[X']$ is connected, hence X' is a Steiner tree.

If there is no optimum Steiner tree X satisfying $X \subseteq M$, then by applying the previous argument to an optimum Steiner tree, we obtain an optimum Steiner tree X satisfying $|X| = |K| + 1$. □

After recursing until no module $M \in \Pi_{mod}(G)$ contains all terminals (and updating G accordingly), we can apply the following reduction to solve the problem if the quotient graph is prime. Let $(G, K, \mathbf{c}, \bar{b})$ be a Steiner Tree instance such that $|\pi_V(K)| \geq 2$ and $G^q = G/\Pi_{mod}(G)$ is prime. We consider the Steiner Tree instance $(G^q, K^q, \mathbf{c}^q, \bar{b}^q)$ where $K^q = \pi_V(K)$, $\mathbf{c}^q(v^q_M) = \mathbf{c}(K \cap M) = \sum_{v \in K \cap M} \mathbf{c}(v)$ if $K \cap M \neq \emptyset$ and $\mathbf{c}^q(v^q_M) = \min_{v \in M} \mathbf{c}(v)$ otherwise, and $\bar{b}^q = \bar{b}$.

Lemma 6.3.2. *Suppose that $(G, K, \mathbf{c}, \bar{b})$ is a Steiner Tree instance such that no module $M \in \Pi_{mod}(G)$ contains all terminals K and G^q is prime.*

Then, the answer to the Steiner Tree instance $(G, K, \mathbf{c}, \bar{b})$ is positive if and only if the answer to the Steiner Tree instance $(G^q, K^q, \mathbf{c}^q, \bar{b}^q)$ is positive.

Proof. If X is an optimum Steiner tree of $(G, K, \mathbf{c}, \bar{b})$, then we claim that $X^q = \pi_V(X)$ is a Steiner tree of $(G^q, K^q, \mathbf{c}^q, \bar{b}^q)$ with $\mathbf{c}^q(X^q) \leq \mathbf{c}(X)$. We have that $K^q = \pi_V(K)$, so $K \subseteq X$ implies that $K^q \subseteq X^q$. By Lemma 6.1.2, we see that $G^q[X^q]$ is connected as well. By definition of X^q and $\mathbf{c}^q$, we have for all $v_M^q \in X^q$ that $\mathbf{c}^q(v_M^q) \leq \mathbf{c}(X \cap M)$ and hence $\mathbf{c}^q(X^q) \leq \mathbf{c}(X) \leq \bar{b} = \bar{b}^q$.

If X^q is an optimum Steiner tree of $(G^q, K^q, \mathbf{c}^q, \bar{b}^q)$, then we claim that $X = K \cup \{v_M : v_M^q \in X^q, K \cap M = \emptyset\}$, where $v_M = \arg\min_{v \in M} \mathbf{c}(v)$, is a Steiner tree of $(G, K, \mathbf{c}, \bar{b})$ with $\mathbf{c}(X) \leq \mathbf{c}^q(X^q)$. We have that $K \subseteq X$ by definition of X and for the costs we compute that $\mathbf{c}(X) = \mathbf{c}(K) + \mathbf{c}(X \setminus K) = \mathbf{c}^q(K^q) + \mathbf{c}^q(X^q \setminus K^q) = \mathbf{c}^q(X^q) \leq \bar{b}^q = \bar{b}$. Note that X^q satisfies $X^q = \pi_V(X)$ by definition of X. Therefore, Lemma 6.1.2 implies that $G[X]$ is connected and X is a Steiner tree of G. □

Proposition 6.3.3 ([51])**.** *There exists a Monte-Carlo algorithm that given a tree decomposition of width at most k for G solves* Steiner Tree *in time $\mathcal{O}^*(3^k)$. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.*

Proof. The algorithm presented by Cygan et al. [51] can be easily augmented to handle positive vertex costs in this running time under the assumption that the costs $\mathbf{c}(v)$, $v \in V$, are at most polynomial in $|V|$. □

By recursing, applying Proposition 6.3.3 to solve the reduced instance from Lemma 6.3.2, and handling parallel and series nodes, we obtain the following.

Theorem 6.3.4. *There exists a Monte-Carlo algorithm that given a tree decomposition of width at most k for every prime node in the modular decomposition of G solves* Steiner Tree *in time $\mathcal{O}^*(3^k)$. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.*

Proof. If no module $M \in \Pi_{mod}(G)$ contains all terminals K, then we want to invoke Lemma 6.3.2. If G^q is a parallel node, then the answer is trivially no. If G^q is a series node, then $G[K]$ is already connected, but we have assumed that this is not the case. Hence, by Theorem 2.4.7 G^q must be a prime node and we can indeed invoke Lemma 6.3.2, so it suffices to solve the Steiner Tree instance $(G^q, K^q, \mathbf{c}^q, \bar{b}^q)$. By definition of modular-treewidth, we have $\mathrm{tw}(G^q) \leq \text{mod-tw}(G) \leq k$ and we are given a corresponding tree decomposition of G^q. Hence, we can simply run the algorithm of Proposition 6.3.3 and return its result.

If some module $M \in \Pi_{mod}(G)$ contains all terminals v, then due to Lemma 6.3.1 we first compute in polynomial time an optimum Steiner tree X_1 of G subject to $|X_1| = |K| + 1$ by brute force. If $\mathbf{c}(X_1) \leq \bar{b}$, then we answer yes. Otherwise, we repeatedly recurse into the module M until we reach a node $G_*^q = G_*/\Pi_{mod}(G_*)$ in the modular decomposition of G such that no $M_* \in \Pi_{mod}(G_*)$ contains all terminals K. We can then solve the Steiner Tree instance $(G_*, K, \mathbf{c}|_{V(G_*)}, \bar{b})$ like in the first paragraph and return its answer. Note that this recursion can never lead to a G_* with $|V(G_*)| = 1$ as that would imply $|K| = 1$, which contradicts the assumption that $G[K]$ is not connected.

As we call Proposition 6.3.3 at most once, we obtain the same error bound. □

Cygan et al. [50] have shown that STEINER TREE cannot be solved in time $\mathcal{O}^*((3-\varepsilon)^{\mathrm{pw}(G)})$ for some $\varepsilon > 0$, unless SETH fails. Since $\text{mod-tw}(G) \leq \mathrm{tw}(G) \leq \mathrm{pw}(G)$, this shows that the running time of Theorem 6.3.4 is tight.

6.4 Connected Dominating Set Reduction

When solving CONNECTED DOMINATING SET, we assume that G is connected, otherwise the answer is trivially no, and that the costs $\mathbf{c}(v)$, $v \in V$, are at most polynomial in $|V|$. CONNECTED DOMINATING SET can be solved by essentially considering only the first quotient graph. First, we will have to handle some edge cases though. If the first quotient graph $G^q = G^q_V = G/\Pi_{mod}(G)$ contains a *universal* vertex $v^q_M \in V(G^q)$, i.e., $N_{G^q}[v^q_M] = V(G^q)$, then there could be a connected dominating set X of G that is fully contained in M. We search for such a connected dominating set by recursively solving CONNECTED DOMINATING SET on $G[M]$. At some point, we arrive at a graph, where the first quotient graph does not contain a universal vertex, or at the one-vertex graph. In the latter case, the answer is trivial. Otherwise, the structure of connected dominating sets allows us to solve the problem on the quotient graph G^q.

Lemma 6.4.1. *If G contains at least two vertices, then $G^q = G/\Pi_{mod}(G)$ contains a universal vertex if and only if G^q is a clique.*

Proof. The reverse direction is simple: every vertex of a clique is a universal vertex.

For the forward direction, first notice that G^q cannot be a parallel node if G^q contains a universal vertex. Suppose that G^q contains a universal vertex $v^q_{M_0}$. Consider the set $M = V(G) \setminus M_0$ and notice that M has to be a module of G, because $v^q_{M_0}$ is a universal vertex in G^q. If G^q were a prime node, then all modules in $\Pi_{mod}(G)$ are maximal proper modules by Theorem 2.4.7, but $V(G) = M_0 \cup M$ implies that $|\Pi_{mod}(G)| \leq 2$ which contradicts that G^q is prime. Therefore, the only remaining possibility is that G^q is a series node, i.e., G^q is a clique. □

Lemma 6.4.2. *If G^q is a prime node, then no connected dominating set X of G is contained in a single module $M \in \Pi_{mod}(G)$. Furthermore, for any optimum connected dominating set X of G and module $M \in \Pi_{mod}(G)$ it holds that either $X \cap M = \emptyset$ or $X \cap M = \{v_M\}$, where v_M is some vertex of minimum cost in M.*

Proof. By Lemma 6.4.1, G^q cannot contain a universal vertex. Suppose that $X \subseteq M$ for some $M \in \Pi_{mod}(G)$. Since $v^q_M \in V(G^q)$ is not a universal vertex, there exists a module $M' \in \Pi_{mod}(G) \setminus \{M\}$ that is not adjacent to M, hence X cannot dominate the vertices in M' and thus cannot be a connected dominating set.

For the statement about optimum connected dominating sets, suppose that X is a connected dominating set of G and $\mathbf{c}(X \cap M) > \mathbf{c}(v_M) > 0$, where v_M is some vertex of minimum cost in M, for some $M \in \Pi_{mod}(G)$. The set $X' = (X \setminus M) \cup \{v_M\}$ satisfies $\mathbf{c}(X') < \mathbf{c}(X)$ and Lemma 6.1.1 shows that $G[X']$ is connected. Since X is a connected dominating set intersecting at least two modules, there has to be a module $M' \in \Pi_{mod}(G)$ that is adjacent to M and satisfies $X \cap M' \neq \emptyset$. Since $M \neq M'$, there is some $v \in X' \cap M' \neq \emptyset$ which dominates all vertices in M. Hence, X' is a dominating set as well.

Repeatedly applying this argument shows the statement about optimum connected dominating sets. □

Proposition 6.4.3 ([51]). *There exists an algorithm that given a tree decomposition of width at most k for G and a weight function $\mathbf{w}$ isolating the optimum connected dominating sets solves* Connected Dominating Set *in time $\mathcal{O}^*(4^k)$. If $\mathbf{w}$ is not isolating, then the algorithm may return false negatives.*

Proof. The algorithm presented by Cygan et al. [51] can be easily augmented to handle positive vertex costs in this running time under the assumption that the costs $\mathbf{c}(v)$, $v \in V$, are at most polynomial in $|V|$. Notice that the only source of randomness in the algorithm of Cygan et al. is the sampling of a weight function. Given an isolating weight function, the algorithm will always succeed. □

As for Steiner Tree, the strategy is again to essentially just call the known algorithm for Connected Dominating Set[treewidth] on the quotient graphs. However, a single call will not be sufficient in the case of Connected Dominating Set; to still obtain the same success probability, we will analyze the behavior of isolating weight functions under the following reduction.

Let $(G, \mathbf{c}, \bar{b})$ be a Connected Dominating Set instance such that G^q is a prime node and let $\mathbf{w}\colon V \to \mathbb{N}$ be a weight function. In each $M \in \Pi_{mod}(G)$ pick a vertex $v_M^{\mathbf{c},\mathbf{w}}$ that lexicographically minimizes $(\mathbf{c}(v), \mathbf{w}(v))$ among all vertices $v \in M$. We construct the Connected Dominating Set instance $(G^q, \mathbf{c}^q, \bar{b})$ with $\mathbf{c}^q(v_M^q) = \mathbf{c}(v_M^{\mathbf{c},\mathbf{w}})$ for all $v_M^q \in V(G^q)$ and define the weight function $\mathbf{w}^q(v_M^q) = \mathbf{w}(v_M^{\mathbf{c},\mathbf{w}})$ for all $v_M^q \in V(G^q)$. The following lemma shows how the two Connected Dominating Set instances are related.

Lemma 6.4.4. *Let $(G, \mathbf{c}, \bar{b})$ be a* Connected Dominating Set *instance such that G^q is a prime node, let $\mathbf{w}\colon V \to \mathbb{N}$ be a weight function, and let $(G^q, \mathbf{c}^q, \bar{b})$ and $\mathbf{w}^q$ be defined as above. The following statements hold:*

1. *If X is an optimum connected dominating set of $(G, \mathbf{c})$, then $X^q = \pi_V(X)$ is a connected dominating set of G^q with $\mathbf{c}^q(X^q) = \mathbf{c}(X)$.*
2. *If X^q is an optimum connected dominating set of $(G^q, \mathbf{c}^q)$, then $X = \{v_M^{\mathbf{c},\mathbf{w}} : v_M^q \in X^q\}$ is a connected dominating set of G with $\mathbf{c}(X) = \mathbf{c}^q(X^q)$.*
3. *If $\mathbf{w}$ isolates the optimum connected dominating sets of $(G, \mathbf{c})$, then $\mathbf{w}^q$ isolates the optimum connected dominating sets of $(G^q, \mathbf{c}^q)$.*

Proof. First, notice that the subgraph $G' = (V', E')$ of G induced by $\{v_M^{\mathbf{c},\mathbf{w}} : M \in \Pi_{mod}(G)\}$ is isomorphic to G^q.

1. Let X be an optimum connected dominating set of $(G, \mathbf{c})$ and set $X^q = \pi_V(X)$. We compute

$$\mathbf{c}^q(X^q) = \sum_{v_M^q \in X^q} \mathbf{c}^q(v_M^q) = \sum_{\substack{M \in \Pi_{mod}(G):\\ X \cap M \neq \emptyset}} \mathbf{c}(v_M^{\mathbf{c},\mathbf{w}}) = \sum_{\substack{M \in \Pi_{mod}(G):\\ X \cap M \neq \emptyset}} \mathbf{c}(X \cap M) = \mathbf{c}(X),$$

   where the penultimate equality follows from Lemma 6.4.2 and the choice of $v_M^{\mathbf{c},\mathbf{w}}$. Furthermore, we can assume $X \cap M = \{v_M^{\mathbf{c},\mathbf{w}}\}$ whenever $X \cap M \neq \emptyset$ by Lemma 6.4.2.

Then, the isomorphism between G^q and G' also maps X^q to X, and hence X^q has to be a connected dominating set of G^q.

2. Suppose that X^q is an optimum connected dominating set of $(G^q, \mathbf{c}^q)$. Defining X as above, we see that X^q satisfies $X^q = \pi_V(X)$. By Lemma 6.4.1, G^q contains no universal vertex, hence $|X^q| \geq 2$ and X must intersect at least two modules. Therefore, Lemma 6.1.2 shows that $G[X]$ is connected. The isomorphism between G^q and G' shows that X must dominate all vertices in V'.
   For any vertex $v \in V \setminus (X \cup V')$ and its module $v \in M \in \Pi_{mod}(G)$, we claim that there exists a module $M' \in \Pi_{mod}(G)$ such that $v_{M'}^{\mathbf{c},\mathbf{w}} \in X$ dominates v. If $X \cap M = \emptyset$, then there exists an adjacent module M' with $X \cap M' \neq \emptyset$, because the vertex $v_M^{\mathbf{c},\mathbf{w}} \in V'$ must be dominated by X. If $X \cap M \neq \emptyset$, a module M' with the same properties exists, because X intersects at least two modules and $G[X]$ is connected. In either case, $v_{M'}^{\mathbf{c},\mathbf{w}}$ must dominate the vertex v by the module property, hence X is a connected dominating set of G. It remains to compute

$$\mathbf{c}(X) = \sum_{v_M^{\mathbf{c},\mathbf{w}} \in X} \mathbf{c}(v_M^{\mathbf{c},\mathbf{w}}) = \sum_{v_M^q \in X^q} \mathbf{c}^q(v_M^q) = \mathbf{c}^q(X^q).$$

3. The first two statements show that connected dominating sets in $(G, \mathbf{c})$ and $(G^q, \mathbf{c}^q)$ have the same optimum cost. Suppose that $\mathbf{w}$ is a weight function that isolates the optimum connected dominating sets of $(G, \mathbf{c})$ and let X be the optimum connected dominating set that is isolated by $\mathbf{w}$. Therefore, X lexicographically minimizes $(\mathbf{c}(X), \mathbf{w}(X))$ among all connected dominating sets of G. By Lemma 6.4.2, we know that $X \cap M = \{v'_M\}$ whenever $X \cap M \neq \emptyset$, where v'_M is a vertex of minimum cost in M.
   We claim that $v'_M = v_M^{\mathbf{c},\mathbf{w}}$ for all modules $M \in \Pi_{mod}(G)$ with $X \cap M \neq \emptyset$. By definition of $v_M^{\mathbf{c},\mathbf{w}}$, we must have $\mathbf{w}(v'_M) \geq \mathbf{w}(v_M^{\mathbf{c},\mathbf{w}})$. If $\mathbf{w}(v'_M) > \mathbf{w}(v_M^{\mathbf{c},\mathbf{w}})$, then we could reduce the weight of X by exchanging v'_M with $v_M^{\mathbf{c},\mathbf{w}}$, contradicting the minimality of $(\mathbf{c}(X), \mathbf{w}(X))$. If $\mathbf{w}(v'_M) = \mathbf{w}(v_M^{\mathbf{c},\mathbf{w}})$ and $v'_M \neq v_M^{\mathbf{c},\mathbf{w}}$, then X cannot be the isolated connected dominating set, as exchanging v'_M and $v_M^{\mathbf{c},\mathbf{w}}$ yields a connected dominating set of the same cost and weight. This proves the claim.
   Using the claim, we compute

$$\mathbf{w}^q(X^q) = \sum_{v_M^q \in X^q} \mathbf{w}^q(v_M^q) = \sum_{\substack{M \in \Pi_{mod}(G): \\ X \cap M \neq \emptyset}} \mathbf{w}(v_M^{\mathbf{c},\mathbf{w}}) = \mathbf{w}(X).$$

   Finally, consider any other optimum connected dominating set $Y^q \neq X^q$ of G^q. Setting $Y = \{v_M^{\mathbf{c},\mathbf{w}} : v_M^q \in Y^q\} \neq X$, we obtain $Y^q = \pi_V(Y)$ and $\mathbf{c}(Y) = \mathbf{c}^q(Y^q) = \mathbf{c}^q(X^q) = \mathbf{c}(X)$, hence $\mathbf{w}^q(Y^q) = \mathbf{w}(Y) > \mathbf{w}(X) = \mathbf{w}^q(X^q)$, where the inequality follows since $\mathbf{w}$ isolates the optimum connected dominating sets of $(G, \mathbf{c})$. Thus, $\mathbf{w}^q$ isolates the optimum connected dominating sets of $(G^q, \mathbf{c}^q)$. □

Theorem 6.4.5. *There exists a Monte-Carlo algorithm that given a tree decomposition of width at most k for every prime node in the modular decomposition of G solves* Connected Dominating Set *in time $\mathcal{O}^*(4^k)$. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.*

Proof. We begin by sampling a weight function $\mathbf{w}\colon V \to [2|V|]$. By Lemma 3.1.4, $\mathbf{w}$ isolates the optimum connected dominating sets of $(G, \mathbf{c})$ with probability at least $1/2$. The algorithm proceeds top-down through the modular decomposition tree of G, but we only recurse further if the current node is a series node. Each recursive call is determined by some $M^\uparrow \in \mathcal{M}_{\text{tree}}(G)$ and we have to determine in this call if a connected dominating set X of $G[M^\uparrow]$ with $\mathbf{c}(X) \leq \bar{b}$ exists, i.e., solve the Connected Dominating Set instance $(G[M^\uparrow], \mathbf{c}|_{M^\uparrow}, \bar{b})$. The weight function $\mathbf{w}$ is passed along by considering its restriction, i.e., $\mathbf{w}|_{M^\uparrow}$.

Let $\mathbb{A}_{\text{tw}}$ denote the algorithm from Proposition 6.4.3. Our algorithm may perform several calls to $\mathbb{A}_{\text{tw}}$, where each call may return false negatives when the considered weight function is not isolating. We return to the error analysis after finishing the description of the modular-treewidth algorithm.

We begin by explaining the three base cases. If $|M^\uparrow| = 1$, then we let $M^\uparrow = \{v_{M^\uparrow}\}$ and check whether $\mathbf{c}(v_{M^\uparrow}) \leq \bar{b}$ and return yes or no accordingly. Otherwise, we have $|M^\uparrow| \geq 2$ and can consider $G^q_{M^\uparrow}$. If $G^q_{M^\uparrow}$ is a parallel node, then the answer is trivially no. If $G^q_{M^\uparrow}$ is a prime node, then we can invoke Lemma 6.4.4 to reduce the Connected Dominating Set instance $(G[M^\uparrow], \mathbf{c}|_{M^\uparrow}, \bar{b})$ to a Connected Dominating Set instance on the quotient graph $G^q_{M^\uparrow}$. We are given a tree decomposition of $G^q_{M^\uparrow}$ of width at most k by assumption. We run $\mathbb{A}_{\text{tw}}$ on the quotient instance together with the weight function from Lemma 6.4.4 and return its result.

Finally, suppose that $G^q_{M^\uparrow}$ is a series node. In this case, any set X of size 2 that intersects two different modules $M \in \texttt{children}(M^\uparrow) = \Pi_{mod}(G[M^\uparrow])$ is a connected dominating set of $G[M^\uparrow]$. We compute all those sets by brute force in polynomial time and return yes if any of them satisfies $\mathbf{c}(X) \leq \bar{b}$. Otherwise, we need to recurse into the modules $M \in \texttt{children}(M^\uparrow)$, because any connected dominating set of $G[M]$ will also be a connected dominating set of $G[M^\uparrow]$. We return true if at least one of these recursive calls returns true. This concludes the description of the algorithm and we proceed with the error analysis now.

The only source of errors is that we may call $\mathbb{A}_{\text{tw}}$ with a non-isolating weight function, but this can only yield false negatives and hence the modular-treewidth algorithm cannot give false positives either. Even if the sampled weight function is isolating, this may not be the case for the restrictions $\mathbf{w}|_{M^\uparrow}$, $M^\uparrow \in \mathcal{M}_{\text{tree}}(G)$. Nonetheless, we show that if $\mathbf{w}$ is isolating, then the modular-treewidth algorithm does not return an erroneous result. To do so, we show that if $\mathbf{w}|_{M^\uparrow}$ is isolating at a series node, then the weight function in the branch containing the isolated optimum connected dominating set must be isolating as well.

To be precise, suppose that $G^q_{M^\uparrow}$ is a series node and that $\mathbf{w}|_{M^\uparrow}$ isolates X^* among the optimum connected dominating sets of $(G[M^\uparrow], \mathbf{c}|_{M^\uparrow})$. We claim that $\mathbf{w}|_M$, $M \in \texttt{children}(M^\uparrow)$, isolates X^* among the optimum connected dominating sets of $(G[M], \mathbf{c}|_M)$ if $X^* \subseteq M$. This follows by a simple exchange argument: if $\mathbf{w}|_M$ is not isolating, i.e., there is some optimum connected dominating set $X \neq X^*$ of $(G[M], \mathbf{c}|_M)$ with $\mathbf{w}(X) = \mathbf{w}(X^*)$, then X is also an optimum connected dominating set of $(G[M^\uparrow], \mathbf{c}|_{M^\uparrow})$, contradicting that $\mathbf{w}|_{M^\uparrow}$ is isolating X^*. If X^* intersects multiple modules $M \in \texttt{children}(M^\uparrow)$, then X^* is found deterministically among the sets of size 2.

As $\mathbf{w}$ is isolating with probability at least $1/2$ this concludes the error analysis. Furthermore, for every module $M \in \mathcal{M}_{\text{tree}}(G)$, we need at most time $\mathcal{O}^*(4^k)$. Therefore, the theorem statement follows. □

Cygan et al. [50] have shown that Connected Dominating Set cannot be solved in time $\mathcal{O}^*((4-\varepsilon)^{\text{pw}(G)})$ for some $\varepsilon > 0$, unless SETH fails. Since $\text{mod-tw}(G) \leq \text{tw}(G) \leq \text{pw}(G)$, this shows that the running time of Theorem 6.4.5 is tight.

6.5 Connected Vertex Cover Algorithm

In the Connected Vertex Cover problem, we are given a graph $G = (V, E)$, a cost function $\mathbf{c}\colon V \to \mathbb{N} \setminus \{0\}$, and an integer $\bar{b}$ and we have to decide whether there exists a subset of vertices $X \subseteq V$ with $\mathbf{c}(X) \leq \bar{b}$ such that $G - X$ contains no edges and $G[X]$ is connected. We will assume that the values of the cost function $\mathbf{c}$ are polynomially bounded in the size of the graph G. We also assume that G is connected and contains at least two vertices, hence $|\Pi_{mod}(G)| \geq 2$ and $G^q := G^q_V = G/\Pi_{mod}(G)$ cannot be edgeless.

To solve Connected Vertex Cover, we begin by computing some optimum (possibly non-connected) vertex cover Y_M with respect to $\mathbf{c}|_M$ for every module $M \in \Pi_{mod}(G)$ that $G[M]$ contains at least one edge. If $G[M]$ contains no edges, then we set $Y_M = \{v^*_M\}$, where $v^*_M \in M$ is a vertex minimizing the cost inside M, i.e., $v^*_M := \arg\min_{v \in M} \mathbf{c}(v)$. The vertex covers can be computed in time $\mathcal{O}^*(2^{\text{mod-tw}(G)})$ by using the algorithm from Theorem 6.2.3.

Definition 6.5.1. Let $X \subseteq V$ be a vertex subset. We say that X is *nice* if for every module $M \in \Pi_{mod}(G)$ it holds that $X \cap M \in \{\emptyset, Y_M, M\}$.

We will show that it is sufficient to only consider nice vertex covers via some exchange arguments. This allows us to only consider a constant number of states per module in the dynamic programming algorithm.

Lemma 6.5.2. *If there exists a connected vertex cover X of G that intersects at least two modules in $\Pi_{mod}(G)$, then there exists a connected vertex cover X' of G that is nice and intersects at least two modules in $\Pi_{mod}(G)$ with $\mathbf{c}(X') \leq \mathbf{c}(X)$.*

Proof. Let X be the given connected vertex cover. Via exchange arguments, we will see that we can find a nice connected vertex cover with the same cost. Suppose that there is a module $M \in \Pi_{mod}(G)$ such that $G[M]$ contains no edges and $1 \leq |X \cap M| < |M|$. We claim that $X' = (X \setminus M) \cup \{v^*_M\}$ is a connected vertex cover with $\mathbf{c}(X') \leq \mathbf{c}(X)$. For any module $M' \in \Pi_{mod}(G)$ adjacent to M, we must have that $X' \cap M' = X \cap M' = M'$, else there would be an edge between M and M' that is not covered by X. In particular, all edges incident to M are already covered by $X \setminus M = X' \setminus M$. By Lemma 6.1.1, X' is connected and we have that $\mathbf{c}(X') \leq \mathbf{c}(X)$ due to the choice of v^*_M.

If $M \in \Pi_{mod}(G)$ is a module such that $G[M]$ contains at least one edge, then we consider two cases. If $\mathbf{c}(X \cap M) < \mathbf{c}(Y_M)$, then $X \cap M$ cannot be a vertex cover of $G[M]$ and hence X would not be a vertex cover of G. If $\mathbf{c}(Y_M) \leq \mathbf{c}(X \cap M) < \mathbf{c}(M)$, then we claim that $X' = (X \setminus M) \cup Y_M$ is a connected vertex cover with $\mathbf{c}(X') \leq \mathbf{c}(X)$. By assumption, we have $\mathbf{c}(X') \leq \mathbf{c}(X)$. We must have that $X \cap M \neq M$, therefore, as before, X and X' must

fully contain all modules adjacent to M to cover all edges leaving M. Since $G[M]$ contains at least one edge, we have that $Y_M \neq \emptyset$ and $G[X']$ must be connected by Lemma 6.1.1.

By repeatedly applying these arguments to X, we obtain the claim. □

The next lemma enables us to handle connected vertex covers that are contained in a single module with polynomial-time preprocessing.

Lemma 6.5.3. *A vertex set $X \subseteq V$ is a connected vertex cover of G with $X \subseteq M$ for some module $M \in \Pi_{mod}(G)$ if and only if $X = M$, all edges of G are incident to M, and $G[M]$ is connected.*

Proof. The reverse direction is trivial. We will show the forward direction. Since G is connected and $|\Pi_{mod}(G)| \geq 2$, there exists a module $M' \in \Pi_{mod}(G)$ adjacent to M. If $X \neq M$, then there exists an edge between M and M' that is not covered by X. If there is an edge in G not incident to M, then clearly X cannot cover all edges. Clearly, $G[X] = G[M]$ must be connected. □

Before going into the main algorithm, we handle the edge case of series nodes. The following lemma shows that there are only a polynomial number of interesting cases for series nodes, hence we can check them by brute force in polynomial time.

Lemma 6.5.4. *If G^q is a clique of size at least two, then for any vertex cover X there is some $M' \in \Pi_{mod}(G)$ such that for all other modules $M' \neq M \in \Pi_{mod}(G)$, we have $X \cap M = M$.*

Proof. Suppose there are two modules $M_1 \neq M_2 \in \Pi_{mod}(G)$ such that $X \cap M_1 \neq M_1$ and $X \cap M_2 \neq M_2$. These modules are adjacent, because G^q is a clique and thus X cannot be a vertex cover, since there exists an uncovered edge between $M_1 \setminus X$ and $M_2 \setminus X$. □

6.5.1 Dynamic Programming for Prime Nodes

It remains to handle the case that G is a prime node. Due to Lemma 6.5.3, we only need to look for connected vertex covers that intersect at least two modules in $\Pi_{mod}(G)$ now. Hence, we can make use of Lemma 6.5.2 and Lemma 6.1.4. We are given a tree decomposition $(\mathcal{T}^q, (B^q_t)_{t \in V(\mathcal{T}^q)})$ of the quotient graph $G^q := G^q_V = G/\Pi_{mod}(G)$ of width k and by Lemma 2.4.2, we can assume that it is a very nice tree decomposition.

To solve Connected Vertex Cover on G, we perform dynamic programming along the tree decomposition $\mathcal{T}^q$ using the cut-and-count-technique. Lemma 6.1.4 will allow us to work directly on the quotient graph. We begin by presenting the cut-and-count-formulation of the problem. For any subgraph G' of G, we define the *relaxed solutions* $\mathcal{R}(G') = \{X \subseteq V(G') : X \text{ is a nice vertex cover of } G'\}$ and *the cut solutions* $\mathcal{Q}(G') = \{(X, (X_L, X_R)) \in \mathcal{C}^{hom}_V(G') : X \in \mathcal{R}(G')\}$.

For the isolation lemma, cf. Lemma 3.1.4, we sample a weight function $\mathbf{w} \colon V \to [2n]$ uniformly at random. We will need to track the cost $\mathbf{c}(X)$, the weight $\mathbf{w}(X)$, and the number of intersected modules $|\pi_V(X)|$ of each partial solution $(X, (X_L, X_R))$. Accordingly, we define $\mathcal{R}^{\overline{c},\overline{w},\overline{m}}(G') = \{X \in \mathcal{R}(G') : \mathbf{c}(X) = \overline{c}, \mathbf{w}(X) = \overline{w}, |\pi_V(X)| = \overline{m}\}$ and $\mathcal{Q}^{\overline{c},\overline{w},\overline{m}}(G') = \{(X, (X_L, X_R)) \in \mathcal{Q}(G') : X \in \mathcal{R}^{\overline{c},\overline{w},\overline{m}}(G')\}$ for all subgraphs G' of G, $\overline{c} \in [0, \mathbf{c}(V)], \overline{w} \in [0, \mathbf{w}(V)], \overline{m} \in [0, |\Pi_{mod}(G)|]$.

As discussed, to every node $t \in V(\mathcal{T}^q)$ we associate a subgraph $G_t^q = (V_t^q, E_t^q)$ of G^q in the standard way, which in turn gives rise to a subgraph $G_t = (V_t, E_t)$ of G. The subgraphs G_t grow module by module and are considered by the dynamic program, hence we define $\mathcal{R}_t^{\bar{c},\overline{w},\overline{m}} = \mathcal{R}^{\bar{c},\overline{w},\overline{m}}(G_t)$ and $\mathcal{Q}_t^{\bar{c},\overline{w},\overline{m}} = \mathcal{Q}^{\bar{c},\overline{w},\overline{m}}(G_t)$ for all $\bar{c}$, $\overline{w}$, and $\overline{m}$. We will compute the sizes of the sets $\mathcal{Q}_t^{\bar{c},\overline{w},\overline{m}}$ by dynamic programming over the tree decomposition $\mathcal{T}^q$, but to do so we need to parameterize the partial solutions by their state on the current bag.

Disregarding the side of the cut, Lemma 6.5.2 tells us that each module $M \in \Pi_{mod}(G)$ has one of three possible states for some $X \in \mathcal{R}_t^{\bar{c},\overline{w},\overline{m}}$, namely $X \cap M \in \{\emptyset, Y_M, M\}$. Since we are considering homogeneous cuts there are two possibilities if $X \cap M \neq \emptyset$; $X \cap M$ is contained in the left side of the cut or in the right side. Thus, there are five total choices. We define $\mathbf{States} = \{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R, \mathbf{A}_L, \mathbf{A}_R\}$ with $\mathbf{1}$ denoting that the partial solution contains at least one vertex, but not all, from the module and with $\mathbf{A}$ denoting that the partial solution contains all vertices of the module; the subscript denotes the side of the cut.

A function of the form $f\colon B_t^q \to \mathbf{States}$ is called *t-signature*. For every node $t \in V(\mathcal{T}^q)$, cost $\bar{c} \in [0, \mathbf{c}(V)]$, weight $\overline{w} \in [0, \mathbf{w}(V)]$, number of modules $\overline{m} \in [0, |\Pi_{mod}(G)|]$, and t-signature f, the family $\mathcal{A}_t^{\bar{c},\overline{w},\overline{m}}(f)$ consists of all $(X, (X_L, X_R)) \in \mathcal{Q}_t^{\bar{c},\overline{w},\overline{m}}$ that satisfy for all $v_M^q \in B_t^q$:

$$
\begin{aligned}
&f(v_M^q) = \mathbf{0} \quad \leftrightarrow \quad X \cap M = \emptyset, \\
&f(v_M^q) = \mathbf{1}_L \leftrightarrow X_L \cap M = Y_M \neq M, && f(v_M^q) = \mathbf{1}_R \leftrightarrow X_R \cap M = Y_M \neq M, \\
&f(v_M^q) = \mathbf{A}_L \leftrightarrow X_L \cap M = M, && f(v_M^q) = \mathbf{A}_R \leftrightarrow X_R \cap M = M.
\end{aligned}
$$

Recall that by considering homogeneous cuts, we have that $X_L \cap M = \emptyset$ or $X_R \cap M = \emptyset$ for every module $M \in \Pi_{mod}(G)$. We use the condition $Y_M \neq M$ for the states $\mathbf{1}_L$ and $\mathbf{1}_R$ to ensure a well-defined state for modules of size 1. Note that the sets $\mathcal{A}_t^{\bar{c},\overline{w},\overline{m}}(f)$, ranging over f, partition $\mathcal{Q}_t^{\bar{c},\overline{w},\overline{m}}$ due to considering nice vertex covers and homogeneous cuts.

Our goal is to compute the size of $\mathcal{A}_{\hat{r}}^{\bar{c},\overline{w},\overline{m}}(\emptyset) = \mathcal{Q}_{\hat{r}}^{\bar{c},\overline{w},\overline{m}} = \mathcal{Q}^{\bar{c},\overline{w},\overline{m}}(G)$, where $\hat{r}$ is the root vertex of the tree decomposition $\mathcal{T}^q$, modulo 4 for all $\bar{c}$, $\overline{w}$, $\overline{m}$. By Lemma 6.1.4, there is a connected vertex cover X of G with $\mathbf{c}(X) = \bar{c}$ and $\mathbf{w}(X) = \overline{w}$ if the result is nonzero.

We present the recurrences for the various bag types to compute $A_t^{\bar{c},\overline{w},\overline{m}}(f) = |\mathcal{A}_t^{\bar{c},\overline{w},\overline{m}}(f)|$; if not stated otherwise, then $t \in V(\mathcal{T}^q)$, $\bar{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$, $\overline{m} \in [0, |\Pi_{mod}(G)|]$, and f is a t-signature. We set $A_t^{\bar{c},\overline{w},\overline{m}}(f) = 0$ whenever at least one of $\bar{c}$, $\overline{w}$, or $\overline{m}$ is negative.

Leaf Bag. We have that $B_t^q = B_t = \emptyset$ and t has no children. The only possible t-signature is $\emptyset$ and the only possible partial solution is $(\emptyset, (\emptyset, \emptyset))$. Hence, we only need to check the tracker values:

$$A_t^{\bar{c},\overline{w},\overline{m}}(\emptyset) = [\bar{c} = 0][\overline{w} = 0][\overline{m} = 0].$$

Introduce Vertex Bag. We have $B_t^q = B_s^q \cup \{v_M^q\}$, where $s \in V(\mathcal{T}^q)$ is the only child of t and $v_M^q \notin B_s^q$. Hence, $B_t = B_s \cup M$. We have to consider all possible interactions of a partial solution with M, since we are considering nice vertex covers these interactions are quite restricted. To formulate the recurrence, we let, as an exceptional case, f be an s-signature here and not a t-signature. Since no edges of the quotient graph G^q incident

to v_M^q are introduced yet, we only have to check some edge cases and update the trackers when introducing v_M^q:

$$\begin{array}{lcll} A_t^{\overline{c},\overline{w},\overline{m}}(f[v_M^q \mapsto \mathbf{0}]) & = & [G[M] \text{ is edgeless}] & A_s^{\overline{c},\overline{w},\overline{m}}(f), \\ A_t^{\overline{c},\overline{w},\overline{m}}(f[v_M^q \mapsto \mathbf{1}_L]) & = & [|M| > 1] & A_s^{\overline{c}-\mathbf{c}(Y_M),\overline{w}-\mathbf{w}(Y_M),\overline{m}-1}(f), \\ A_t^{\overline{c},\overline{w},\overline{m}}(f[v_M^q \mapsto \mathbf{1}_R]) & = & [|M| > 1] & A_s^{\overline{c}-\mathbf{c}(Y_M),\overline{w}-\mathbf{w}(Y_M),\overline{m}-1}(f), \\ A_t^{\overline{c},\overline{w},\overline{m}}(f[v_M^q \mapsto \mathbf{A}_L]) & = & & A_s^{\overline{c}-\mathbf{c}(M),\overline{w}-\mathbf{w}(M),\overline{m}-1}(f), \\ A_t^{\overline{c},\overline{w},\overline{m}}(f[v_M^q \mapsto \mathbf{A}_R]) & = & & A_s^{\overline{c}-\mathbf{c}(M),\overline{w}-\mathbf{w}(M),\overline{m}-1}(f). \end{array}$$

Introduce Edge Bag. Let $\{v_{M_1}^q, v_{M_2}^q\}$ denote the introduced edge. We have that $\{v_{M_1}^q, v_{M_2}^q\} \subseteq B_t^q = B_s^q$. The edge $\{v_{M_1}^q, v_{M_2}^q\}$ corresponds to adding a join between the modules M_1 and M_2. We need to filter all solutions whose states at M_1 and M_2 are not consistent with M_1 and M_2 being adjacent. There are essentially two possible reasons: either not all edges between M_1 and M_2 are covered, or the introduced edges go across the homogeneous cut. We implement this via the helper function $\mathrm{feas}\colon \mathbf{States} \times \mathbf{States} \to \{0,1\}$ which is defined by $\mathrm{feas}(\mathbf{s}_1, \mathbf{s}_2) = [\{\mathbf{s}_1, \mathbf{s}_2\} \cap \{\mathbf{A}_L, \mathbf{A}_R\} \neq \emptyset][\mathbf{s}_1 \in \{\mathbf{1}_L, \mathbf{A}_L\} \to \mathbf{s}_2 \notin \{\mathbf{1}_R, \mathbf{A}_R\}][\mathbf{s}_1 \in \{\mathbf{1}_R, \mathbf{A}_R\} \to \mathbf{s}_2 \notin \{\mathbf{1}_L, \mathbf{A}_L\}]$ or, equivalently, the following table:

feas	$\mathbf{0}$	$\mathbf{1}_L$	$\mathbf{1}_R$	$\mathbf{A}_L$	$\mathbf{A}_R$
$\mathbf{0}$	0	0	0	1	1
$\mathbf{1}_L$	0	0	0	1	0
$\mathbf{1}_R$	0	0	0	0	1
$\mathbf{A}_L$	1	1	0	1	0
$\mathbf{A}_R$	1	0	1	0	1

The recurrence is then simply given by

$$A_t^{\overline{c},\overline{w},\overline{m}}(f) = \mathrm{feas}(f(v_{M_1}^q), f(v_{M_2}^q)) A_s^{\overline{c},\overline{w},\overline{m}}(f).$$

Forget Vertex Bag. We have that $B_t^q = B_s^q \setminus \{v_M^q\}$, where $v_M^q \in B_s^q$ and $s \in V(\mathcal{T}^q)$ is the only child of t. Here, we only need to forget the state at v_M^q and accumulate the contributions from the different states v_M^q could assume, as the states are disjoint no overcounting happens:

$$A_t^{\overline{c},\overline{w},\overline{m}}(f) = \sum_{\mathbf{s} \in \mathbf{States}} A_s^{\overline{c},\overline{w},\overline{m}}(f[v \mapsto \mathbf{s}]).$$

Join Bag. We have $B_t^q = B_{s_1}^q = B_{s_2}^q$, where $s_1, s_2 \in V(\mathcal{T}^q)$ are the children of t. Two partial solutions, one at s_1, and the other at s_2, can be combined when the states agree on all $v_M^q \in B_t^q$. Since we update the trackers already at introduce vertex bags, we need to take care that the values of the modules in the bag are not counted twice. For this sake, define $S^f = \bigcup_{v_M^q \in f^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\})} Y_M \cup \bigcup_{v_M^q \in f^{-1}(\{\mathbf{A}_L, \mathbf{A}_R\})} M$ for all t-signatures f. This definition satisfies $X \cap B_t = S^f$ for all $(X, (X_L, X_R)) \in \mathcal{A}_t^{\overline{c},\overline{w},\overline{m}}(f)$. Then, the recurrence is given by

$$A_t^{\overline{c},\overline{w},\overline{m}}(f) = \sum_{\substack{\overline{c}_1 + \overline{c}_2 = \overline{c} + \mathbf{c}(S^f) \\ \overline{w}_1 + \overline{w}_2 = \overline{w} + \mathbf{w}(S^f)}} \sum_{\overline{m}_1 + \overline{m}_2 = \overline{m} + (|B_t^q| - f^{-1}(\mathbf{0}))} A_{s_1}^{\overline{c}_1,\overline{w}_1,\overline{m}_1}(f) A_{s_2}^{\overline{c}_2,\overline{w}_2,\overline{m}_2}(f).$$

Lemma 6.5.5. *If G^q is prime, then there exists a Monte-Carlo algorithm that, given a tree decomposition for G^q of width at most k and the sets Y_M for all $M \in \Pi_{mod}(G)$, determines whether there is a connected vertex cover X of G with $\mathbf{c}(X) \leq \overline{b}$ intersecting at least two modules of $\Pi_{mod}(G)$ in time $\mathcal{O}^*(5^k)$. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.*

Proof. The algorithm samples a weight function $\mathbf{w}\colon V \to [2n]$ uniformly at random. Using the recurrences, we compute the values $A_{\hat{r}}^{\overline{c},\overline{w},\overline{m}}(\emptyset)$ modulo 4 for all $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$, $\overline{m} \in [2, |\Pi_{mod}(G)|]$. Setting $\mathcal{S}^{\overline{c},\overline{w},\overline{m}} = \{X \in \mathcal{R}^{\overline{c},\overline{w},\overline{m}}(G) : G[X] \text{ is connected}\}$, we have that

$$|\mathcal{Q}^{\overline{c},\overline{w},\overline{m}}(G)| = |\mathcal{Q}_{\hat{r}}^{\overline{c},\overline{w},\overline{m}}| = A_{\hat{r}}^{\overline{c},\overline{w},\overline{m}}(\emptyset) = \sum_{X \in \mathcal{R}^{\overline{c},\overline{w},\overline{m}}(G)} 2^{\mathtt{cc}(G[X])} \equiv_4 2|\mathcal{S}^{\overline{c},\overline{w},\overline{m}}|$$

by Lemma 6.1.4. By Lemma 3.1.4, $\mathbf{w}$ isolates the set of optimum nice connected vertex covers intersecting at least two modules of $\Pi_{mod}(G)$ with probability at least $1/2$. If $\overline{c}$ denotes the optimum value, then there exist choices of $\overline{w}$ and $\overline{m}$ such that $|\mathcal{S}^{\overline{c},\overline{w},\overline{m}}| = 1$ and hence $A_{\hat{r}}^{\overline{c},\overline{w},\overline{m}}(\emptyset) \not\equiv_4 0$. The algorithm searches for the smallest such $\overline{c}$ and returns true if $\overline{c} \leq \overline{b}$. Note that if a connected vertex cover X intersecting at least two modules with $\mathbf{c}(X) \leq \overline{b}$ exists, then so does a nice one by Lemma 6.5.2. If $\overline{c} > \overline{b}$, the algorithm returns false.

Next, we argue that the running time is $\mathcal{O}^*(5^k)$. Since a very nice tree decomposition has polynomially many nodes and since the cost function $\mathbf{c}$ is assumed to be polynomially bounded, there are $\mathcal{O}^*(5^k)$ table entries to compute. Furthermore, it is easy to see that every recurrence can be computed in polynomial time. It remains to prove the correctness of the provided recurrences.

If t is a leaf node, then $V_t = \emptyset$ and hence $\mathcal{Q}_t^{\overline{c},\overline{w},\overline{m}}$ can contain at most $(\emptyset, (\emptyset, \emptyset))$, and we have that $\mathbf{c}(\emptyset) = \mathbf{w}(\emptyset) = |\pi_V(\emptyset)| = 0$, which is checked by the recurrence.

If t is an introduce vertex node introducing v_M^q, consider $(X, (X_L, X_R)) \in \mathcal{A}_t^{\overline{c},\overline{w},\overline{m}}(f[v_M^q \mapsto \mathbf{s}])$, where f is some s-signature and $\mathbf{s} \in \mathbf{States}$. We have that $(X \setminus M, (X_L \setminus M, X_R \setminus M)) \in \mathcal{A}_s^{\overline{c}',\overline{w}',\overline{m}'}(f)$ for $\overline{c}' = \mathbf{c}(X \setminus M)$, $\overline{w}' = \mathbf{w}(X \setminus M)$, $\overline{m}' = |\pi_V(X \setminus M)|$. Depending on $\mathbf{s}$, we argue that this sets up a bijection between $\mathcal{A}_t^{\overline{c},\overline{w},\overline{m}}(f[v_M^q \mapsto \mathbf{s}])$ and $\mathcal{A}_s^{\overline{c}',\overline{w}',\overline{m}'}(f)$. The injectivity of this map follows in general by observing that $\mathbf{s}$ completely determines the interaction of $(X, (X_L, X_R))$ with M.

- $\mathbf{s} = \mathbf{0}$: We have $X \cap M = \emptyset$, which implies that $G[M]$ does not contain an edge, as X cannot be a vertex cover of G_t otherwise. In this case, the mapping is essentially the identity mapping, so it is bijective and the trackers do not change.
- $\mathbf{s} = \mathbf{1}_L$: We have $X \cap M = X_L \cap M = Y_M \neq M$ and $X_R \cap M = \emptyset$. Due to $\emptyset \neq Y_M \neq M$, we have that $|M| > 1$. As $X \cap M = Y_M$, we update the trackers according to Y_M. Note that any $(X', (X'_L, X'_R)) \in \mathcal{A}_s^{\overline{c}',\overline{w}',\overline{m}'}(f)$ is hit by $(X' \cup Y_M, (X'_L \cup Y_M, X'_R)) \in \mathcal{A}_t^{\overline{c},\overline{w},\overline{m}}(f[v_M^q \mapsto \mathbf{s}])$, which relies on the fact that no edges incident to v_M^q have been introduced yet, so that neither the vertex cover property nor consistent cut property can be violated when extending by Y_M.
- $\mathbf{s} = \mathbf{1}_R$: analogous to the previous case.

- $\mathbf{s} = \mathbf{A}_L$: We have $X \cap M = X_L \cap M = M$ and $X_R \cap M = \emptyset$. Hence, we update the trackers according to M. For surjectivity, we see that $(X', (X'_L, X'_R)) \in \mathcal{A}_s^{\bar{c}', \overline{w}', \overline{m}'}(f)$ is hit by $(X' \cup M, (X'_L \cup M, X'_R)) \in \mathcal{A}_t^{\bar{c}, \overline{w}, \overline{m}}(f[v_M^q \mapsto \mathbf{s}])$, which again relies on the fact that no edges incident to v_M^q have been introduced yet.
- $\mathbf{s} = \mathbf{A}_R$: analogous to the previous case.

If t is an introduce edge bag introducing edge $\{v_{M_1}^q, v_{M_2}^q\}$, then $\mathcal{Q}_t^{\bar{c}, \overline{w}, \overline{m}} \subseteq \mathcal{Q}_s^{\bar{c}, \overline{w}, \overline{m}}$ and we filter out all $(X, (X_L, X_R)) \in \mathcal{Q}_s^{\bar{c}, \overline{w}, \overline{m}} \setminus \mathcal{Q}_t^{\bar{c}, \overline{w}, \overline{m}}$. A partial solution $(X, (X_L, X_R)) \in \mathcal{Q}_s^{\bar{c}, \overline{w}, \overline{m}}$ must be filtered if and only if an edge between M_1 and M_2 is not covered or an edge between $X \cap M_1$ and $X \cap M_2$ connects both sides of the homogeneous cut. These criteria are implemented by feas; the first case corresponds to $\mathrm{feas}(\mathbf{s}_1, \mathbf{s}_2) = \mathbf{0}$ for all $\mathbf{s}_1, \mathbf{s}_2 \in \{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$ and the second corresponds to $\mathrm{feas}(\mathbf{s}_1, \mathbf{s}_2) = \mathbf{0}$ whenever $\mathbf{s}_1 \neq \mathbf{0} \neq \mathbf{s}_2$ and the cut subscript of $\mathbf{s}_1$ and $\mathbf{s}_2$ disagrees.

If t is a forget vertex bag forgetting v_M^q, then $\mathcal{Q}_t^{\bar{c}, \overline{w}, \overline{m}} = \mathcal{Q}_s^{\bar{c}, \overline{w}, \overline{m}}$ and every $(X, (X_L, X_R)) \in \mathcal{Q}_t^{\bar{c}, \overline{w}, \overline{m}}$ is counted by some $A_s^{\bar{c}, \overline{w}, \overline{m}}(f[v_M^q \mapsto \mathbf{s}])$ with $\mathbf{s}$ being the appropriate state and the states are disjoint as already noted.

If t is a join bag, then $V_t = V_{s_1} \cup V_{s_2}$ and $B_t = B_{s_1} = B_{s_2} = V_{s_1} \cap V_{s_2}$. Since G_{s_1} and G_{s_2} are subgraphs of G_t, any $(X, (X_L, X_R)) \in \mathcal{A}_t^{\bar{c}, \overline{w}, \overline{m}}(f)$ splits into $(X^1, (X_L^1, X_R^1)) \in \mathcal{A}_{s_1}^{\bar{c}_1, \overline{w}_1, \overline{m}_1}(f)$ and $(X^2, (X_L^2, X_R^2)) \in \mathcal{A}_{s_2}^{\bar{c}_2, \overline{w}_2, \overline{m}_2}(f)$, where $X^i = X \cap V_{s_i}$, $X_L^i = X_L \cap V_{s_i}$, $X_R^i = X_R \cap V_{s_i}$ for $i \in [2]$. Since $S^f = X \cap B_t = X^1 \cap B_t = X^2 \cap B_t$, some overcounting occurs when adding up e.g. the costs $\bar{c}_1$ and $\bar{c}_2$. This is accounted for by the equation $\bar{c}_1 + \bar{c}_2 = \bar{c} + \mathbf{c}(S^f)$ and similarly for the weights and the number of modules hit by X. Vice versa, the union of the graphs G_{s_1} and G_{s_2} yields G_t, and any $(X^1, (X_L^1, X_R^1)) \in \mathcal{A}_{s_1}^{\bar{c}_1, \overline{w}_1, \overline{m}_1}(f)$ and $(X^2, (X_L^2, X_R^2)) \in \mathcal{A}_{s_2}^{\bar{c}_2, \overline{w}_2, \overline{m}_2}(f)$ must agree on B_t, since the behavior on B_t is completely specified by f. Therefore, one can argue that $(X^1 \cup X^2, (X_L^1 \cup X_L^2, X_R^1 \cup X_R^2)) \in \mathcal{A}_t^{\bar{c}, \overline{w}, \overline{m}}(f)$. □

Putting everything together, we obtain the following algorithm.

Theorem 6.5.6. *There exists a Monte-Carlo algorithm that given a tree decomposition of width at most k for every prime quotient graph $H \in \mathcal{H}_p(G)$, solves* Connected Vertex Cover *in time $\mathcal{O}^*(5^k)$. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.*

Proof. If $|V(G)| = 1$, then $\emptyset$ is a connected vertex cover and we can always answer true. Otherwise, we first compute the sets Y_M for all $M \in \Pi_{mod}(G)$ in time $\mathcal{O}^*(2^k)$ using Theorem 6.2.3. Using Lemma 6.5.3, we first check in polynomial time if there is any connected vertex cover X of G contained in a single module with $\mathbf{c}(X) \leq \bar{b}$. If yes, then we return true. Otherwise, we will proceed based on the node type of $V(G)$ in the modular decomposition of G.

If $V(G)$ is a parallel node, i.e., G^q is an independent set of size at least two, then G cannot be connected, contradicting our assumption. If $V(G)$ is a series node, i.e., G^q is a clique of size at least two, then we solve the problem in polynomial time using Lemma 6.5.2 and Lemma 6.5.4, which tell us that there only $3|\Pi_{mod}(G)|$ possible solutions to consider.

If G^q is prime, then it remains to search for connected vertex covers intersecting at least two modules and hence we can invoke Lemma 6.5.5. □

Note that Theorem 6.5.6 gets a tree decomposition for *every* quotient graph as input, whereas Lemma 6.5.5 only requires a tree decomposition for the topmost quotient graph. This is due to the fact that the algorithm in Theorem 6.2.3 to compute the vertex cover Y_M of $G[M]$ for every $M \in \mathcal{M}_{\text{tree}}(G)$, requires a decomposition for every quotient graph, but the vertex covers are enough information to enable us to solve Connected Vertex Cover by just considering the topmost quotient graph.

6.6 Feedback Vertex Set Algorithm

The cut-and-count-technique applies more naturally to the dual problem Induced Forest instead of Feedback Vertex Set, so we choose to study the dual problem. An instance of Induced Forest consists of a graph $G = (V, E)$, and a budget $\bar{b} \in \mathbb{N}$, and the task is to decide whether there exists a vertex set $X \subseteq V$ with $|X| \geq \bar{b}$ such that $G[X]$ is a forest. As our algorithm is quite technical, we only consider the case of unit costs here to reduce the number of technical details.

For Connected Vertex Cover, it was sufficient to essentially only look at the first quotient graph, because we did not have to compute *connected* vertex covers for the subproblems, only usual vertex covers. However, for Induced Forest this is not the case; here, we do need to compute an induced forest in each module $M \in \mathcal{M}_{\text{tree}}(G)$. This essentially means that we need a *nested* dynamic programming algorithm; one *outer dynamic program (outer DP)* along the modular decomposition tree and one *inner dynamic program (inner DP)* along the tree decompositions of the quotient graphs solving the subproblems of the outer DP.

The inner dynamic programming algorithm will again be using the cut-and-count-technique and can therefore produce erroneous results due to the randomization. We will carefully analyze where errors can occur and see that a single global sampling of an isolating weight function will be sufficient, even though some subproblems corresponding to nodes of the modular decomposition tree might be solved incorrectly[2]. For this reason, the notation in this section will more closely track which node of the modular decomposition we are working on, as the setup in the Connected Vertex Cover algorithm would be too obfuscating here.

Notation. $M^{\uparrow} \in \mathcal{M}^*_{\text{tree}}(G)$ will denote the parent module and represents the current subproblem to be solved by the inner DP. The inner DP works on the quotient graph $G^q_{M^{\uparrow}} = G[M^{\uparrow}]/\Pi_{mod}(G[M^{\uparrow}])$ whose vertices correspond to modules $M \in \texttt{children}(M^{\uparrow}) = \Pi_{mod}(G[M^{\uparrow}])$; associated to the quotient graph $G^q_{M^{\uparrow}}$ is the projection $\pi_{M^{\uparrow}} \colon M^{\uparrow} \to V(G^q_{M^{\uparrow}})$. By $v^q_M \in G^q_{M^{\uparrow}}$ we refer to the vertex in the quotient graph corresponding to M. At times, it will be useful to not have to specify the parent module and we say that two modules $M_1, M_2 \in \mathcal{M}_{\text{tree}}(G)$ are *siblings* if there is some $M^{\uparrow} \in \mathcal{M}_{\text{tree}}(G)$ such that $M_1, M_2 \in \texttt{children}(M^{\uparrow})$, i.e., they have the same parent. For a module $M \in \mathcal{M}_{\text{tree}}(G)$, let $\mathcal{N}_{\texttt{sib}}(M)$

[2]This more involved analysis can be avoided by using larger weights for the isolation lemma, but this will lead to an additional factor of $\mathcal{O}(n)$ in the running time: by using weights of size $\mathcal{O}(n^2)$ each subproblem is solved correctly with probability $\Omega(1/n)$ and, since there are $\mathcal{O}(n)$ subproblems, a simple union bound shows that all subproblems are solved correctly with constant error probability.

denote the family of sibling modules of M that are adjacent to M and we define $\mathcal{N}_{\texttt{all}}(M) = \{M' \in \mathcal{M}_{\text{tree}}(G) : M \cap M' = \emptyset, E_G(M, M') \neq \emptyset\}$, i.e., the family of all strong modules that are adjacent to M.

6.6.1 Structure of Optimum Induced Forests

We begin by studying the structure of optimum induced forests with respect to the modular decomposition. Let $\mathcal{F}_{opt}(G)$ be the family of maximum induced forests of G. We start by giving some definitions to capture the structure of induced forests with respect to the modular decomposition.

Definition 6.6.1. Let $X \subseteq V(G)$ be a vertex subset. We associate with X a *module-marking* $\varphi_X \colon \mathcal{M}_{\text{tree}}(G) \to \{\mathbf{0}, \mathbf{1}, \mathbf{2}_{\mathcal{I}}, \mathbf{2}_{\mathcal{E}}\}$ defined by

$$\varphi_X(M) = \begin{cases} \mathbf{0}, & \text{if } |X \cap M| = 0, \\ \mathbf{1}, & \text{if } |X \cap M| = 1, \\ \mathbf{2}_{\mathcal{I}}, & \text{if } |X \cap M| \geq 2 \text{ and } G[X \cap M] \text{ contains no edge}, \\ \mathbf{2}_{\mathcal{E}}, & \text{if } |X \cap M| \geq 2 \text{ and } G[X \cap M] \text{ contains at least one edge}. \end{cases}$$

We use module-markings to describe the states taken by an induced forest X on the modules $M \in \mathcal{M}_{\text{tree}}(G)$. Ordering $\mathbf{0} < \mathbf{1} < \mathbf{2}_{\mathcal{I}} < \mathbf{2}_{\mathcal{E}}$, note that every module-marking φ_X is *monotone* in the following sense: for all $M_1, M_2 \in \mathcal{M}_{\text{tree}}(G)$ the inclusion $M_1 \subseteq M_2$ implies that $\varphi_X(M_1) \leq \varphi_X(M_2)$.

Any induced forest has to satisfy some local properties relative to the modules which are captured by the following definition.

Definition 6.6.2. Let $X \subseteq V(G)$ be a vertex subset. We say that X is *forest-nice* if for every $M \in \mathcal{M}_{\text{tree}}(G)$ the following properties hold:

- If $\varphi_X(M) = \mathbf{2}_{\mathcal{I}}$, then $\varphi_X(\mathcal{N}_{\texttt{all}}(M)) \subseteq \{\mathbf{0}, \mathbf{1}\}$ and $|\mathcal{N}_{\texttt{sib}}(M) \cap \varphi_X^{-1}(\mathbf{1})| \leq 1$.
- If $\varphi_X(M) = \mathbf{2}_{\mathcal{E}}$, then $\varphi_X(\mathcal{N}_{\texttt{all}}(M)) \subseteq \{\mathbf{0}\}$.

The "degree-condition" $|\mathcal{N}_{\texttt{sib}}(M) \cap \varphi_X^{-1}(\mathbf{1})| \leq 1$ deliberately only talks about the sibling modules, as we can have arbitrarily long chains of modules with $v \in M_1 \subseteq M_2 \subseteq \cdots \subseteq M_\ell$, so no useful statement is possible if we would instead consider all modules.

Lemma 6.6.3. *Every induced forest $X \subseteq V(G)$ of G is forest-nice.*

Proof. Consider any $M \in \mathcal{M}_{\text{tree}}(G)$ with $|X \cap M| \geq 2$. If there were some module $M' \in \mathcal{N}_{\texttt{all}}(M)$ with $|X \cap M'| \geq 2$, then $G[X \cap (M \cup M')]$ contains a cycle of size 4 as all edges between M and M' exist in G, hence such M' cannot exist. If, additionally, $G[X \cap M]$ contains an edge, then any $M' \in \mathcal{N}_{\texttt{all}}(M)$ with $X \cap M' \neq \emptyset$ would necessarily lead to a cycle of size 3 in $G[X \cap (M \cup M')]$, hence such M' cannot exist. Finally, suppose that $\varphi_X(M) = \mathbf{2}_{\mathcal{I}}$ and two neighboring sibling modules $M_1 \neq M_2 \in \mathcal{N}_{\texttt{sib}}(M)$ with $\varphi_X(M_1) = \varphi_X(M_2) = \mathbf{1}$ exist. We must have $M_1 \cap M_2 = \emptyset$ and therefore a cycle of size 4 would exist in $G[X \cap (M \cup M_1 \cup M_2)]$, which is again not possible. □

The modular structure allows us to perform the following exchange arguments.

Lemma 6.6.4. *Let X be an induced forest of G and $M \in \mathcal{M}_{\text{tree}}(G)$.*

1. *If $\varphi_X(M) = \mathbf{2}_{\mathcal{I}}$ and Y is an independent set of $G[M]$, then $(X \setminus M) \cup Y$ is an induced forest of G.*
2. *If $\varphi_X(M) = \mathbf{2}_{\mathcal{E}}$ and Y is an induced forest of $G[M]$, then $(X \setminus M) \cup Y$ is an induced forest of G.*

Proof. We set $X' = (X \setminus M) \cup Y$ in both cases. Since $X' \setminus M = X \setminus M$, there cannot be any cycle in $G[X' \setminus M]$. Also there cannot be any cycle in $G[X \cap M] = G[Y]$ by assumption.

1. Suppose there is a cycle C' in $G[X']$. By the previous arguments, we must have $C' \cap M \neq \emptyset$ and $C' \setminus M \neq \emptyset$. We will argue that such a cycle would give rise to a cycle C in $G[X]$, contradicting the assumption that X is an induced forest. Let $v_1, \ldots, v_\ell, v_1$ be the sequence of vertices visited by C' and let $v_{i_1}, \ldots, v_{i_r}$ with $1 \leq i_1 < \cdots < i_r \leq \ell$ denote the vertices of C' that are in M. If some edge of C', say $\{v_1, v_2\}$ without loss of generality, is contained in $G[X' \setminus M]$, pick some $u \in X \cap M$ and consider the cycle C given by the vertex sequence $v_1, v_2, \ldots, v_{i_1-1}, u, v_{i_r+1}, \ldots, v_\ell, v_1$; C is a cycle of $G[X]$ as the edges $\{v_{i_1-1}, u\}$ and $\{u, v_{i_r+1}\}$ exist in G, because $u, v_{i_1-1}, v_{i_r+1} \in M$. If no such edge exists in C', then C' is a cycle in the biclique with parts $X' \cap M$ and $N_G(X' \cap M)$, in particular $|C' \cap M| \geq 2$ and $|C' \setminus M| \geq 2$. Since $|X \cap M| \geq 2$ by assumption and $|X \cap M| = |X' \cap M| \geq |C' \setminus M| \geq 2$, it follows that $G[X]$ contains a biclique with parts of size at least two and hence $G[X]$ must contain a cycle.
2. Since X is forest-nice by Lemma 6.6.3, $\varphi_X(M) = \mathbf{2}_{\mathcal{E}}$ implies the inclusion $\varphi_{X'}(\mathcal{N}_{\mathtt{all}}(M)) = \varphi_X(\mathcal{N}_{\mathtt{all}}(M)) \subseteq \{\mathbf{0}\}$, and therefore $(X' \cap M, X' \setminus M)$ is a consistent cut of $G[X']$. Therefore any cycle C in $G[X']$ must be fully contained in either $X' \cap M$ or $X' \setminus M$, but we ruled out each of these cases previously. Hence, $G[X']$ contains no cycle. □

Lemma 6.6.4 allows us to see that maximum induced forests must make locally optimal choices inside each module. We capture these local choices with the following two definitions.

Definition 6.6.5. Let $X \subseteq V(G)$ be a vertex subset. We say that X has *optimal substructure* if for every $M \in \mathcal{M}_{\text{tree}}(G)$ the following properties hold:

- If $\varphi_X(M) = \mathbf{2}_{\mathcal{I}}$, then $X \cap M$ is a maximum independent set of $G[M]$.
- If $\varphi_X(M) = \mathbf{2}_{\mathcal{E}}$, then $X \cap M$ is a maximum induced forest of $G[M]$.

Definition 6.6.6. Let $X \subseteq V(G)$ be a vertex subset. We say that X has the *promotion property* if for every $M \in \mathcal{M}_{\text{tree}}(G)$ with $|X \cap M| \geq 2$ and $\varphi_X(\mathcal{N}_{\mathtt{all}}(M)) = \{\mathbf{0}\}$, we have that $X \cap M$ is a maximum induced forest of $G[M]$.

While we could have subsumed the promotion property as part of the definition of optimal substructure, we define it separately as it has more involved implications on the dynamic program and deserves separate care.

Lemma 6.6.7. *Every maximum induced forest of G, i.e., $X \in \mathcal{F}_{opt}(G)$, has optimal substructure and the promotion property.*

Proof. Lemma 6.6.3 already shows that X is forest-nice. If X would not have optimal substructure, then we can invoke Lemma 6.6.4 to obtain a larger induced forest X', hence X would not be a maximum induced forest.

We prove a strengthened exchange argument to show the promotion property. We claim that for any induced forest X of G, module $M \in \mathcal{M}_{\text{tree}}(G)$ with $\varphi_X(M) \in \{\mathbf{2}_{\mathcal{I}}, \mathbf{2}_{\mathcal{E}}\}$ and $\varphi_X(\mathcal{N}_{\mathtt{all}}(M)) \subseteq \{\mathbf{0}\}$, and induced forest Y of $G[M]$, the set $X' = (X \setminus M) \cup Y$ is again an induced forest of G. Suppose that $G[X']$ contains a cycle C'. By assumption on X, C' cannot be contained in $G[X' \setminus M] = G[X \setminus M]$. By assumption on Y, C' cannot be contained in $G[X' \cap M] = G[Y]$. Therefore, C' must intersect $X' \cap M$ and $X' \setminus M$ simultaneously. However, $\varphi_{X'}(\mathcal{N}_{\mathtt{all}}(M)) = \varphi_X(\mathcal{N}_{\mathtt{all}}(M)) \subseteq \{\mathbf{0}\}$ implies that $(X' \cap M, X' \setminus M)$ is a consistent cut of $G[X']$ and hence such a cycle C' cannot exist. Therefore X' is also an induced forest. If an induced forest X violates the promotion property, then we can invoke this exchange argument to see that X cannot be a maximum induced forest. □

Since any induced forest X is forest-nice, the condition $\varphi_X(M) = \mathbf{2}_{\mathcal{E}}$ implies that $\varphi_X(\mathcal{N}_{\mathtt{all}}(M)) \subseteq \{\mathbf{0}\}$ and therefore the second condition of optimal substructure also follows from the promotion property.

The requirement $|X \cap M| \geq 2$ in the promotion property could also be removed. However, the dynamic programming on quotient graphs will only apply the underlying exchange argument when $|X \cap M| \geq 2$ holds, therefore we already add this requirement here.

Note that a forest-nice vertex subset X does not necessarily induce a forest as a cycle could be induced by the modules $M \in \Pi_{mod}(G)$ with $\varphi_X(M) = \mathbf{1}$.

6.6.2 Application of Isolation Lemma

We will again use the cut-and-count-technique and the isolation lemma to solve Induced Forest[modular-treewidth]. However, since Induced Forest is a maximization problem, we feel it is more natural to use a maximization version of the isolation lemma as we must closely investigate when isolation transfers to subproblems. Let us define the appropriate terminology.

Definition 6.6.8. A function $\mathbf{w}\colon U \to \mathbb{Z}$ *max-isolates* a set family $\mathcal{F} \subseteq 2^U$ if there is a unique $S' \in \mathcal{F}$ with $\mathbf{w}(S') = \max_{S \in \mathcal{F}} \mathbf{w}(S)$, where for subsets X of U we define $\mathbf{w}(X) = \sum_{u \in X} \mathbf{w}(u)$.

Lemma 6.6.9 (Adapt proof of [134] or [167])**.** *Let $\mathcal{F} \subseteq 2^U$ be a nonempty set family over a universe U. Let $N \in \mathbb{N}$ and for each $u \in U$ choose a weight $\mathbf{w}(u) \in [N]$ uniformly and independently at random. Then $\mathbb{P}[\mathbf{w}$ max-isolates $\mathcal{F}] \geq 1 - |U|/N$.*

Due to Lemma 6.6.3 and Lemma 6.6.7, we want our algorithm to compute maximum independent sets and maximum induced forests of $G[M]$ for every $M \in \mathcal{M}_{\text{tree}}(G)$. The computation of the maximum independent sets can be done deterministically quickly enough using Theorem 6.2.3. To compute the maximum induced forests however, we essentially want to recursively call our algorithm again, but the algorithm is randomized. Doing this naively and sampling a weight function for each call would exponentially decrease the success probability depending on the depth of the modular decomposition tree.

To circumvent this issue, we sample a global weight function only once and let the subproblems inherit this weight function, observing that for all "important" subproblems the

inherited weight function is max-isolating if the global weight function is (for appropriate choices of set families).

We define $\mathcal{F}_{opt}(G, \mathbf{s})$, where $\mathbf{s} \in \{\mathbf{0}, \mathbf{1}, \mathbf{2}_{\mathcal{I}}, \mathbf{2}_{\mathcal{E}}\}$, as the family of maximum sets X subject to $G[X]$ being a forest and $\varphi_X(V(G)) \leq \mathbf{s}$. Hence, we have that $\mathcal{F}_{opt}(G, \mathbf{2}_{\mathcal{E}}) = \mathcal{F}_{opt}(G)$ and $\mathcal{F}_{opt}(G, \mathbf{2}_{\mathcal{I}})$ is the family of maximum independent sets of G and $\mathcal{F}_{opt}(G, \mathbf{1})$ is the family of singleton sets.

Lemma 6.6.10. *Let $N \in \mathbb{N}$ and assume that $\mathbf{w}\colon V(G) \to [N]$ is a weight function that max-isolates $\mathcal{F}_{opt}(G)$. Let $X \in \mathcal{F}_{opt}(G)$ be the set that is max-isolated by $\mathbf{w}$. For every $M \in \mathcal{M}_{\text{tree}}(G)$, we have that $\mathbf{w}\big|_M$ max-isolates $X \cap M$ in $\mathcal{F}_{opt}(G[M], \varphi_X(M))$.*

Proof. X has optimal substructure due to Lemma 6.6.7, therefore we have $X \cap M \in \mathcal{F}_{opt}(G[M], \varphi_X(M))$ for all $M \in \mathcal{M}_{\text{tree}}(G)$. Suppose there is some $M \in \mathcal{M}_{\text{tree}}(G)$ such that $\mathbf{w}\big|_M$ does not max-isolate $\mathcal{F}_{opt}(G[M], \varphi_X(M))$, then there is some $X \cap M \neq Y \in \mathcal{F}_{opt}(G[M], \varphi_X(M))$ with $\mathbf{w}(Y) \geq \mathbf{w}(X \cap M)$. By Lemma 6.6.4, $X' = (X \setminus M) \cup Y$ must satisfy $X' \in \mathcal{F}_{opt}(G)$, $X' \neq X$, and $\mathbf{w}(X') \geq \mathbf{w}(X)$. However, then $\mathbf{w}$ cannot max-isolate X in $\mathcal{F}_{opt}(G)$. □

We remark that the previous lemma allows for the possibility that, e.g. $\mathbf{w}\big|_M$ max-isolates $\mathcal{F}_{opt}(G[M], \mathbf{2}_{\mathcal{I}})$, but $\mathbf{w}\big|_M$ does not max-isolate $\mathcal{F}_{opt}(G[M], \mathbf{2}_{\mathcal{E}}) = \mathcal{F}_{opt}(G[M])$, which can lead to our algorithm not finding an optimum induced forest for this subinstance.

6.6.3 Detecting Acyclicness

Let us describe how to check whether a forest-nice subset X induces a forest. The property of being forest-nice essentially allows us to only consider the induced subset on a quotient graph which we then handle by lifting cut-and-count. The property of being forest-nice is a global property in the sense that it considers the whole modular decomposition tree. We first introduce a local version of forest-nice that only considers the children of a parent module $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$:

Definition 6.6.11. Let $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$, $\widetilde{G}^q$ be a subgraph of $G^q_{M^\uparrow}$, and $X \subseteq M^\uparrow$ with $X^q := \pi_{M^\uparrow}(X) \subseteq V(\widetilde{G}^q)$, we say that X is *$M^\uparrow$-forest-nice with respect to $\widetilde{G}^q$*, if the following properties hold for all $v^q_M \in V(\widetilde{G}^q)$:

- If $\varphi_X(M) = \mathbf{2}_{\mathcal{I}}$, then $\deg_{\widetilde{G}^q[X^q]}(v^q_M) \leq 1$ and $\varphi_X(M') \in \{\mathbf{0}, \mathbf{1}\}$ for all $v^q_{M'} \in N_{\widetilde{G}^q}(v^q_M)$.
- If $\varphi_X(M) = \mathbf{2}_{\mathcal{E}}$, then $\varphi_X(M') = \mathbf{0}$ for all $v^q_{M'} \in N_{\widetilde{G}^q}(v^q_M)$.

In the case $\widetilde{G}^q = G^q_{M^\uparrow}$, we simply say that X is *$M^\uparrow$-forest-nice*.

As the (very nice) tree decomposition of $G^q_{M^\uparrow}$ adds edges one-by-one, we need to account for changes in the neighborhoods of vertices in the local definition of forest-niceness via $\widetilde{G}^q$. Otherwise, Definition 6.6.11 is essentially the same definition as Definition 6.6.2, but only considering the child modules of $M^\uparrow$. In particular, if X is forest-nice, then $X \cap M^\uparrow$ is $M^\uparrow$-forest-nice for all $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$.

The next lemma essentially shows that in a $M^\uparrow$-forest-nice set X no cycles intersecting some module $M \in \texttt{children}(M^\uparrow)$ in more than one vertex exist, hence all possible cycles can already be seen in the quotient graph.

Lemma 6.6.12. *Let $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$ and $X \subseteq M^\uparrow$ be $M^\uparrow$-forest-nice and suppose that $G[X \cap M]$ is a forest for all modules $M \in \texttt{children}(M^\uparrow)$ and define $X^q = \pi_{M^\uparrow}(X)$. Then, $G[X]$ is a forest if and only if $G^q_{M^\uparrow}[X^q]$ is a forest.*

Proof. The graph $G^q_{M^\uparrow}[X^q]$ can be considered a subgraph of $G[X]$, so if $G^q_{M^\uparrow}[X^q]$ is not a forest, then neither is $G[X]$.

For the other direction, suppose that $G[X]$ contains a cycle C. It cannot be that $C \subseteq X \cap M$ for some $M \in \texttt{children}(M^\uparrow)$, since $G[X \cap M]$ contains no cycle by assumption. It also cannot be that $G[C \cap M]$ contains an edge for some $M \in \texttt{children}(M^\uparrow)$, since $M^\uparrow$-forest-nice would then imply that C is contained in M, which we just ruled out. If $|C \cap M| \geq 2$ for some $M \in \texttt{children}(M^\uparrow)$, then $M^\uparrow$-forest-nice implies that at most one neighboring sibling module M' is intersected by C and $|C \cap M'| \geq 1$, but since $G[C \cap M]$ cannot contain an edge, this means that the vertices in $C \cap M$ must have degree one in C, so C cannot be a cycle. Finally, we must have $|C \cap M| \leq 1$ for all $M \in \texttt{children}(M^\uparrow)$, but any such cycle C clearly gives rise to a cycle $C^q = \pi_{M^\uparrow}(C)$ in $G^q[X^q]$, too. □

Lemma 6.6.13 (Lemma 4.5 in [51])**.** *Let G be a graph with n vertices and m edges. Then, G is a forest if and only if $\texttt{cc}(G) \leq n - m$ if and only if $\texttt{cc}(G) = n - m$.*

· One could use the *marker technique* already used by Cygan et al. [51] for the treewidth-parameterization together with Lemma 6.6.13 to obtain a cut-and-count algorithm, but the marker technique results in several further technical details to take care of. The marker technique can be avoided by working modulo higher powers of two instead of only modulo two, which was also done by Nederlof et al. [137] when applying cut-and-count to edge-based problems parameterized by treedepth. We also do so, to obtain a cleaner presentation of our algorithm.

Lemma 6.6.14. *Let $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$ and $X \subseteq M^\uparrow$ be $M^\uparrow$-forest-nice and suppose that $G[X \cap M]$ is a forest for all modules $M \in \texttt{children}(M^\uparrow)$. Let $X^q = \pi_{M^\uparrow}(X)$ and let $\overline{v} = |X^q|$ and $\overline{e} = |E(G^q_{M^\uparrow}[X^q])|$. Then, $G[X]$ is a forest if and only if $|\{(X_L, X_R) : (X, (X_L, X_R)) \in \mathcal{C}^{hom}_{M^\uparrow}(G)\}| \not\equiv_{2^{\overline{v}-\overline{e}+1}} 0$.*

Proof. By Lemma 6.1.4, we have that $|\{(X_L, X_R) : (X, (X_L, X_R)) \in \mathcal{C}^{hom}_{M^\uparrow}(G)\}| = 2^{\texttt{cc}(G^q_{M^\uparrow}[X^q])}$. By Lemma 6.6.13, we see that $G^q_{M^\uparrow}[X^q]$ is a forest if and only if $|\{(X_L, X_R) : (X, (X_L, X_R)) \in \mathcal{C}^{hom}_{M^\uparrow}(G)\}| \neq 0 \mod 2^{\overline{v}^q - \overline{e}^q + 1}$. The lemma then follows via Lemma 6.6.12. □

6.6.4 Outer Dynamic Programming Algorithm

Fix an Induced Forest instance $(G = (V, E), \overline{b})$ and a weight function $\mathbf{w}\colon V \to [N]$ throughout this section. To solve Induced Forest[modular-treewidth], we perform dynamic programming in two ways: we proceed bottom-up along the modular decomposition tree of G and to compute the table entries for the node corresponding to module $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$, we use the tables of the children $\texttt{children}(M^\uparrow) = \Pi_{mod}(G[M^\uparrow])$ and perform dynamic programming along the tree decomposition of the associated quotient graph $G^q_{M^\uparrow} = G[M^\uparrow]/\Pi_{mod}(G[M^\uparrow])$. For every module $M \in \mathcal{M}_{\text{tree}}(G)$, we have the following data precomputed:

- a singleton set $Y_M^{\mathbf{1}}$ in M that maximizes $\mathbf{w}(Y_M^{\mathbf{1}})$ and its weight $w_M^{\mathbf{1}} = \mathbf{w}(Y_M^{\mathbf{1}})$,
- a maximum independent set $Y_M^{\mathbf{2}_\mathcal{I}}$ of $G[M]$ that maximizes $\mathbf{w}(Y_M^{\mathbf{2}_\mathcal{I}})$, the size $c_M^{\mathbf{2}_\mathcal{I}} = |Y_M^{\mathbf{2}_\mathcal{I}}|$ and the weight $w_M^{\mathbf{2}_\mathcal{I}} = \mathbf{w}(Y_M^{\mathbf{2}_\mathcal{I}})$ of such an independent set.

The vertex data can clearly be precomputed in polynomial time and the independent set data can be precomputed in time $\mathcal{O}^*(2^{\text{mod-tw}(G)})$ by running the INDEPENDENT SET algorithm from Theorem 6.2.3.

Candidate Forests. We will recursively define for each module $M^\uparrow \in \mathcal{M}_{\text{tree}}(G)$, the $M^\uparrow$-*candidate forest* $Y_{M^\uparrow}^{\mathbf{2}_\mathcal{E}}$ (which depends on the fixed weight function $\mathbf{w}$). Among all induced forests X of $G[M^\uparrow]$ found by the algorithm, the forest $Y_{M^\uparrow}^{\mathbf{2}_\mathcal{E}}$ lexicographically maximizes $(|X|, \mathbf{w}(X))$. Due to the randomization in the cut-and-count-technique however, it can happen that $Y_{M^\uparrow}^{\mathbf{2}_\mathcal{E}}$ is not necessarily a maximum induced forest of $G[M^\uparrow]$. We will see that if we sampled an isolating weight function $\mathbf{w}$, then no errors will occur for the "important" subproblems, hence still allowing us to find a maximum induced forest of the whole graph. The definition of $Y_{M^\uparrow}^{\mathbf{2}_\mathcal{E}}$ is mutually recursive with the definition of the solution family that will be defined afterwards.

Properties of Candidate Forests. We highlight several properties of the candidate forests that are important for the algorithm.

- The base case is given by $Y_{\{v\}}^{\mathbf{2}_\mathcal{E}} = \{v\}$ for all $v \in V(G)$.
- $Y_{M^\uparrow}^{\mathbf{2}_\mathcal{E}}$ is an induced forest of $G[M^\uparrow]$.
- If $G[M^\uparrow]$ contains no edge, then $Y_{M^\uparrow}^{\mathbf{2}_\mathcal{E}} = Y_{M^\uparrow}^{\mathbf{2}_\mathcal{I}}$.
- If $G[M^\uparrow]$ contains an edge, then $|Y_{M^\uparrow}^{\mathbf{2}_\mathcal{E}}| > |Y_{M^\uparrow}^{\mathbf{2}_\mathcal{I}}|$.

Given $Y_M^{\mathbf{2}_\mathcal{E}}$ for all $M \in \texttt{children}(M^\uparrow)$, we can describe how to compute $Y_{M^\uparrow}^{\mathbf{2}_\mathcal{E}}$. This step depends on which kind of node $M^\uparrow$ corresponds to in the modular decomposition. We first handle the degenerate cases of a parallel or series node and then proceed with the much more challenging case of a prime node.

Computing Candidate Forests in Parallel and Series Nodes

If $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$ is a parallel node, i.e., $G^q_{M^\uparrow}$ is an independent set, then Lemma 6.6.3 and Lemma 6.6.7 tell us to simply take a maximum induced forest inside each child module $M \in \texttt{children}(M^\uparrow)$. Hence, we set $Y_{M^\uparrow}^{\mathbf{2}_\mathcal{E}} = \bigcup_{M \in \texttt{children}(M^\uparrow)} Y_M^{\mathbf{2}_\mathcal{E}}$ and accordingly $c_{M^\uparrow}^{\mathbf{2}_\mathcal{E}} = \sum_{M \in \texttt{children}(M^\uparrow)} c_M^{\mathbf{2}_\mathcal{E}}$ and $w_{M^\uparrow}^{\mathbf{2}_\mathcal{E}} = \sum_{M \in \texttt{children}(M^\uparrow)} w_M^{\mathbf{2}_\mathcal{E}}$.

If $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$ is a series node, then we first analyze the structure of maximum induced forests with respect to a series node.

Lemma 6.6.15. *Let $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$ and X be a maximum induced forest of $G[M^\uparrow]$. If $M^\uparrow$ is a series module, i.e., the quotient graph $G^q_{M^\uparrow}$ is a clique, then one of the following statements holds:*

- *$X \subseteq M$ for some $M \in \texttt{children}(M^\uparrow)$ and X is a maximum induced forest of $G[M]$.*
- *$X \subseteq M_1 \cup M_2$ for some $M_1 \neq M_2 \in \texttt{children}(M^\uparrow)$ and $X \cap M_1$ is a maximum independent set of $G[M_1]$ and $|X \cap M_2| = 1$.*

Proof. Suppose that X intersects three different modules in $\texttt{children}(M^\uparrow)$, since they are all adjacent X would induce a triangle. Hence, X can intersect at most two different

modules. By Lemma 6.6.3 and Lemma 6.6.7, X is forest-nice, has optimal substructure and satisfies the promotion property. If X intersects only a single module M, then the first statement follows due to the promotion property. If X intersects two modules, then the second statement follows due to X being forest-nice and optimal substructure. □

Given the maximum independent sets $Y_M^{\mathbf{2}_\mathcal{I}}$ for all $M \in \texttt{children}(M^\uparrow)$, we can in polynomial time compute an optimum induced forest $\tilde{Y}_{M^\uparrow}$ of $G[M^\uparrow]$ subject to the second condition in Lemma 6.6.15. We compare the induced forests $\tilde{Y}_{M^\uparrow}$ and all $Y_M^{\mathbf{2}_\mathcal{E}}$ for all $M \in \texttt{children}(M^\uparrow)$ lexicographically by their cost and weight and, motivated by Lemma 6.6.15, we let $Y_{M^\uparrow}^{\mathbf{2}_\mathcal{E}}$ be the winner of this comparison.

Computing Candidate Forests in Prime Nodes

To compute the $M^\uparrow$-candidate forest $Y_{M^\uparrow}^{\mathbf{2}_\mathcal{E}}$ when $M^\uparrow$ is a prime node, we will use the cut-and-count-technique and dynamic programming along the given tree decomposition of the quotient graph $G_{M^\uparrow}^q$. Before going into the details of the dynamic programming, we will give the necessary formal definitions to describe the partial solutions of the dynamic programming and the subproblem that has to be solved. This will already allow us to define the induced forest $Y_{M^\uparrow}^{\mathbf{2}_\mathcal{E}}$ and prove the correctness of the outer loop involving the modular decomposition. We first introduce some "local" versions of Definition 6.6.5 and Definition 6.6.6.

Definition 6.6.16. Let $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$ and $X \subseteq M^\uparrow$, we say that X has $M^\uparrow$*-substructure* if for all $M \in \texttt{children}(M^\uparrow)$ we have that $\varphi_X(M) \neq \mathbf{0}$ implies $X \cap M = Y_M^{\varphi_X(M)}$.

Comparing the definition of $M^\uparrow$*-substructure* to *optimal substructure*, we see that in $M^\uparrow$-substructure we only consider the child modules and require the choice of a specified vertex, maximum independent set, or induced forest, respectively. Note that due to the previously discussed issue, $Y_M^{\mathbf{2}_\mathcal{E}}$ does not necessarily need to be a maximum induced forest.

Definition 6.6.17. Let $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$ and $X \subseteq M^\uparrow$. We say that X satisfies the $M^\uparrow$*-promotion property* if for all modules $M \in \texttt{children}(M^\uparrow)$ that satisfy $|X \cap M| \geq 2$ and $\varphi_X(\mathcal{N}_{\texttt{sib}}(M)) = \{\mathbf{0}\}$, it holds that $X \cap M = Y_M^{\mathbf{2}_\mathcal{E}}$.

Definition 6.6.17, unlike Definition 6.6.11, does not need to account for the current subgraph of $G_{M^\uparrow}^q$ as promotion is only checked for modules that have already been forgotten by the tree decomposition, i.e., all incident edges have already been added, and for non-introduced modules M, we simply have $X \cap M = \emptyset$.

We can now define the solution family considered by our algorithm.

Definition 6.6.18. The family $\mathcal{R}_{M^\uparrow}$ consists of all $X \subseteq M^\uparrow$ such that X is $M^\uparrow$-forest-nice wrt. $G_{M^\uparrow}^q$, has $M^\uparrow$-substructure, and satisfies the $M^\uparrow$-promotion property. Given $\overline{c} \in [0, |M^\uparrow|]$, $\overline{w} \in [0, \mathbf{w}(M^\uparrow)]$, $\overline{v} \in [0, |\texttt{children}(M^\uparrow)|]$, $\overline{e} \in [0, \overline{v} - 1]$, the family $\mathcal{R}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}}$ consists of all $X \in \mathcal{R}_{M^\uparrow}$ with

- $|X| = \overline{c}$ and $\mathbf{w}(X) = \overline{w}$,
- $|X^q| = \overline{v}$ and $|E(G_{M^\uparrow}^q[X^q])| = \overline{e}$, where $X^q = \pi_{M^\uparrow}(X)$.

We also define $\mathcal{S}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}} = \{X \in \mathcal{R}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}} : G[X] \text{ is a forest}\}$.

By pairing elements of $\mathcal{R}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}}$ with homogeneous cuts, we can use the cut-and-count-technique to decide whether $\mathcal{S}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}}$ is empty or not.

Definition 6.6.19. The family $\mathcal{Q}_{M^\uparrow}$ consists of all $(X,(X_L,X_R)) \in \mathcal{C}_{M^\uparrow}^{hom}(G)$ with $X \in \mathcal{R}_{M^\uparrow}$. Similarly, $\mathcal{Q}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}}$ consists of all $(X,(X_L,X_R)) \in \mathcal{C}_{M^\uparrow}^{hom}(G)$ with $X \in \mathcal{R}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}}$.

The crucial property of $\mathcal{Q}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}}$ is given by the following lemma.

Lemma 6.6.20. *Let $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$. It holds that $|\mathcal{Q}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}}| \equiv_{2^{\overline{v}-\overline{e}+1}} 2^{\overline{v}-\overline{e}}|\mathcal{S}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}}|$.*

Proof. Consider any $X \in \mathcal{R}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}}$ and let $X^q = \pi_{M^\uparrow}(X)$. If $G[X]$ is a forest, then so is $G^q_{M^\uparrow}[X^q]$ by Lemma 6.6.12 and we have that X contributes exactly $2^{\overline{v}-\overline{e}}$ objects to $\mathcal{Q}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}}$ by Lemma 6.1.4 and Lemma 6.6.13. By Lemma 6.6.14, we see that if $G[X]$ is not a forest, then X contributes a multiple of $2^{\overline{v}-\overline{e}+1}$ objects to $\mathcal{Q}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}}$, which therefore cancel. □

From the sets $\mathcal{Q}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}}$ for a fixed $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$, we can finally give the recursive definition of the $M^\uparrow$-candidate forest $Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}}$.

Definition 6.6.21. Let $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$ such that $G^q_{M^\uparrow}$ is prime. The set of *attained cost-weight-pairs* $P_{M^\uparrow}$ consists of all pairs $(\overline{c},\overline{w})$ such that there exist $\overline{v}$ and $\overline{e}$ with $|\mathcal{Q}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}}| \not\equiv_{2^{\overline{v}-\overline{e}+1}} 0$. We denote the lexicographic maximum pair in $P_{M^\uparrow}$ by $(\overline{c}_{max},\overline{w}_{max})$. Lemma 6.6.20 guarantees the existence of an induced forest Y of $G[M^\uparrow]$ with $|Y| = \overline{c}_{max}$ and $\mathbf{w}(Y) = \overline{w}_{max}$. If $\overline{c}_{max} > |Y_{M^\uparrow}^{\mathbf{2}\mathcal{I}}|$, then the *$M^\uparrow$-candidate forest* $Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}}$ is an arbitrary induced forest among these, else we greedily extend $Y_{M^\uparrow}^{\mathbf{2}\mathcal{I}}$ by some vertices, without introducing cycles, to obtain $Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}}$. We set $c_{M^\uparrow}^{\mathbf{2}\mathcal{E}} = |Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}}|$ and $w_{M^\uparrow}^{\mathbf{2}\mathcal{E}} = \mathbf{w}(Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}})$.

The algorithm does not know the exact set $Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}}$, hence no issue is caused by the arbitrary choice, but the algorithm knows the values $c_{M^\uparrow}^{\mathbf{2}\mathcal{E}}$ and $w_{M^\uparrow}^{\mathbf{2}\mathcal{E}}$. The set $Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}}$ is only used for the analysis of the algorithm. We will see that the choice of $Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}}$ is unique when $\mathbf{w}|_{M^\uparrow}$ isolates the optimum induced forests of $G[M^\uparrow]$, else the choice might not be unique. Only in the latter case can $\overline{c}_* \leq |Y_{M^\uparrow}^{\mathbf{2}\mathcal{I}}|$ occur, but since $G^q_{M^\uparrow}$ is prime, the graph $G[M^\uparrow]$ must contain some edges and hence there exists a larger induced forest that is not an independent set.

Note that $Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}}$ is always an induced forest, but $G[Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}}]$ does not necessarily contain an edge, i.e., $Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}}$ may be an independent set or even a single vertex if $G^q_{M^\uparrow}$ is a parallel node or singleton node. This means that for some $X \subseteq V(G)$ with $X \cap M^\uparrow = Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}}$, we only know $\varphi_X(M^\uparrow) \leq \mathbf{2}\mathcal{E}$ and not necessarily $\varphi_X(M^\uparrow) = \mathbf{2}\mathcal{E}$.

The complete outer DP is summarized in Algorithm 2.

Correctness of Outer DP

Assuming an algorithm that computes the values $|\mathcal{Q}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}}|$ for all prime $G^q_{M^\uparrow}$ and all $\overline{c},\overline{w},\overline{v},\overline{e}$, we obtain an algorithm that implicitly computes $Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}}$ for all $M^\uparrow \in \mathcal{M}_{\text{tree}}(G)$ by starting with $Y_{\{v\}}^{\mathbf{2}\mathcal{E}} = \{v\}$ for all $v \in V$ and performs bottom-up dynamic programming along the modular decomposition tree using the appropriate algorithm based on the node type. While the precise set $Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}}$ is not known to the algorithm, it knows the value $c_{M^\uparrow}^{\mathbf{2}\mathcal{E}} = |Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}}|$. The algorithm returns positively if $c_V^{\mathbf{2}\mathcal{E}} \geq \overline{b}$ and negatively otherwise. As we ensure that $Y_{M^\uparrow}^{\mathbf{2}\mathcal{E}}$ is an induced forest for all $M^\uparrow \in \mathcal{M}_{\text{tree}}(G)$, the algorithm does not return false positives. The next lemma concludes the discussion of the outer DP and implies that the algorithm answers correctly assuming that the weight function $\mathbf{w}$ isolates the maximum induced forests of G.

Algorithm 2: Outer DP to compute $Y^{\mathbf{2}_\mathcal{E}}_{M^\uparrow}$.

if $M^\uparrow$ *is a parallel node* **then**
  $Y^{\mathbf{2}_\mathcal{E}}_{M^\uparrow} := \bigcup_{M \in \texttt{children}(M^\uparrow)} Y^{\mathbf{2}_\mathcal{E}}_{M}$;
else if $M^\uparrow$ *is a series node* **then**
  pick Y_1 among all $Y^{\mathbf{2}_\mathcal{E}}_{M}$, $M \in \texttt{children}(M^\uparrow)$, to lex. maximize $(|Y_1|, \mathbf{w}(Y_1))$;
  pick $Y_2 \in \{Y^{\mathbf{2}_\mathcal{I}}_{M_1} \cup Y^{\mathbf{1}}_{M_2} : M_1 \neq M_2 \in \texttt{children}(M^\uparrow)\}$ to lex. max. $(|Y_2|, \mathbf{w}(Y_2))$;
  pick $Y^{\mathbf{2}_\mathcal{E}}_{M^\uparrow}$ as a winner of the lex. comparison $(|Y_1|, \mathbf{w}(Y_1))$ vs $(|Y_2|, \mathbf{w}(Y_2))$;
else
  compute $|\mathcal{Q}^{\overline{c},\overline{w},\overline{v},\overline{e}}_{M^\uparrow}|$ for all $\overline{c}, \overline{w}, \overline{v}, \overline{e}$ using treewidth-based DP;
  construct $P_{M^\uparrow} = \left\{(\overline{c}, \overline{w}) : \text{ there are } \overline{v}, \overline{e} \text{ such that } |\mathcal{Q}^{\overline{c},\overline{w},\overline{v},\overline{e}}_{M^\uparrow}| \not\equiv_{2^{\overline{v}-\overline{e}+1}} 0\right\}$;
  let $(\overline{c}_{max}, \overline{w}_{max}) \in P_{M^\uparrow}$ be the lexicographic maximum;
  if $\overline{c}_{max} > |Y^{\mathbf{2}_\mathcal{I}}_{M^\uparrow}|$ **then**
    pick any $Y^{\mathbf{2}_\mathcal{E}}_{M^\uparrow}$ among induced forests Y of $G[M^\uparrow]$ with $|Y| = \overline{c}_{max}$ and $\mathbf{w}(Y) = \overline{w}_{max}$;
  else
    obtain $Y^{\mathbf{2}_\mathcal{E}}_{M^\uparrow}$ by greedily extending $Y^{\mathbf{2}_\mathcal{I}}_{M^\uparrow}$ by vertices without creating cycles;

Lemma 6.6.22 (Main Correctness Lemma)**.** *Suppose that* $\mathbf{w}$ *max-isolates* X_* *in* $\mathcal{F}_{opt}(G)$*. The following properties hold for all* $M^\uparrow \in \mathcal{M}_{\text{tree}}(G)$*:*

1. $\mathbf{s}(M^\uparrow) := \varphi_{X_*}(M^\uparrow) \neq \mathbf{0}$ *implies that* $X_* \cap M^\uparrow = Y^{\mathbf{s}(M^\uparrow)}_{M^\uparrow}$*, (*$M^\uparrow$*-substructure for all* $M^\uparrow$*)*
2. $\mathbf{s}(M^\uparrow) = \varphi_{X_*}(M^\uparrow) = \mathbf{2}_\mathcal{E}$ *implies that* $Y^{\mathbf{2}_\mathcal{E}}_{M^\uparrow}$ *is a maximum induced forest of* $G[M^\uparrow]$*,*
3. $\mathbf{s}(M^\uparrow) = \varphi_{X_*}(M^\uparrow) = \mathbf{2}_\mathcal{E}$ *implies that* $X_* \cap M^\uparrow \in \mathcal{R}_{M^\uparrow}$*.*

Proof. Notice that for singleton modules only the first property is relevant and is trivially true. By Lemma 6.6.3 and Lemma 6.6.7, X_* is forest-nice, has optimal substructure and the promotion property. By Lemma 6.6.10, it follows that $\mathbf{w}\big|_{M^\uparrow}$ max-isolates $X_* \cap M^\uparrow$ in $\mathcal{F}_{opt}(G, \mathbf{s}(M^\uparrow))$ for all $M^\uparrow \in \mathcal{M}_{\text{tree}}(G)$. Since X_* is forest-nice, $X_* \cap M^\uparrow$ must be $M^\uparrow$-forest-nice for all $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$ as the quotient graph $G^q_{M^\uparrow}$ captures when two sibling modules $M, M' \in \texttt{children}(M^\uparrow)$ are adjacent.

We proceed by proving the first property whenever $\mathbf{s}(M^\uparrow) \neq \mathbf{2}_\mathcal{E}$. Fix some $M^\uparrow$ with $\mathbf{s}(M^\uparrow) \in \{\mathbf{1}, \mathbf{2}_\mathcal{I}\}$. We have $X_* \cap M^\uparrow, Y^{\mathbf{s}(M^\uparrow)}_{M^\uparrow} \in \mathcal{F}_{opt}(G[M^\uparrow], \mathbf{s}(M^\uparrow))$ by optimal substructure and definition. By choice of $Y^{\mathbf{s}(M^\uparrow)}_{M^\uparrow}$, we have that $\mathbf{w}(X_* \cap M^\uparrow) \leq \mathbf{w}(Y^{\mathbf{s}(M^\uparrow)}_{M^\uparrow})$. By max-isolation of $X_* \cap M^\uparrow$ it follows that $\mathbf{w}(X_* \cap M^\uparrow) = \mathbf{w}(Y^{\mathbf{s}(M^\uparrow)}_{M^\uparrow})$ and even $X_* \cap M^\uparrow = Y^{\mathbf{s}(M^\uparrow)}_{M^\uparrow}$.

The remainder of the proof is an induction along the modular decomposition tree, as the base case we consider modules $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$ with $\mathbf{s}(M^\uparrow) = \mathbf{2}_\mathcal{E}$ and $\mathbf{s}(M) \neq \mathbf{2}_\mathcal{E}$ for all $M \in \texttt{children}(M^\uparrow)$. For the base case, we have already shown that $X_* \cap M = Y^{\mathbf{s}(M)}_{M}$ for all $M \in \texttt{children}(M^\uparrow)$, hence $X_* \cap M^\uparrow$ has $M^\uparrow$-substructure in this case.

We continue with the $M^\uparrow$-promotion property in the base case. Suppose it is violated for some $M \in \texttt{children}(M^\uparrow)$, i.e., $\mathbf{s}(\mathcal{N}_{\texttt{sib}}(M)) \subseteq \{\mathbf{0}\}$ and $X_* \cap M = Y^{\mathbf{2}_\mathcal{I}}_{M} \neq Y^{\mathbf{2}_\mathcal{E}}_{M}$ (using $M^\uparrow$-substructure). By definition of $Y^{\mathbf{2}_\mathcal{E}}_{M}$, we have that $Y^{\mathbf{2}_\mathcal{E}}_{M}$ is an induced forest of $G[M]$ and $Y^{\mathbf{2}_\mathcal{E}}_{M} \neq Y^{\mathbf{2}_\mathcal{I}}_{M}$ if and only if $|Y^{\mathbf{2}_\mathcal{E}}_{M}| > |Y^{\mathbf{2}_\mathcal{I}}_{M}|$. We claim that M must also violate the promotion property of X_*. For this it remains to establish that $\mathbf{s}(\mathcal{N}_{\texttt{all}}(M)) \subseteq \{\mathbf{0}\}$. We have $\mathbf{s}(\mathcal{N}_{\texttt{sib}}(M)) \subseteq \{\mathbf{0}\}$ by assumption, this shows that $\mathbf{s}(M') = \mathbf{0}$ for all $M' \in \mathcal{N}_{\texttt{all}}(M)$ with

$M' \subseteq M^\uparrow$. Every module $M' \in \mathcal{N}_{\texttt{all}}(M)$ with $M' \not\subseteq M^\uparrow$ must be disjoint from $M^\uparrow$ and hence $M' \in \mathcal{N}_{\texttt{all}}(M^\uparrow)$ which implies that $\mathbf{s}(M') = \mathbf{0}$ since X_* is forest-nice.

For the base case, we have now established that $X_* \cap M^\uparrow \in \mathcal{R}_{M^\uparrow}$, as we have verified that $X_* \cap M^\uparrow$ is $M^\uparrow$-forest-nice wrt. $G^q_{M^\uparrow}$, has $M^\uparrow$-substructure, and has the $M^\uparrow$-promotion property. We can now proceed by showing the first and second property for the base case when $\mathbf{s}(M^\uparrow) = \mathbf{2}_\mathcal{E}$. Note that the second property follows from the first one by optimal substructure of X_*, so we only have to prove the first property.

If $G^q_{M^\uparrow}$ is a parallel or series node, then the analysis in Section 6.6.4 shows that $Y^{\mathbf{2}_\mathcal{E}}_{M^\uparrow} \in \mathcal{F}_{opt}(G[M^\uparrow])$. Since also $X_* \cap M^\uparrow \in \mathcal{F}_{opt}(G[M^\uparrow])$ and both maximize their weight (by definition and max-isolation), the isolation of $X_* \cap M^\uparrow$ implies $X_* \cap M^\uparrow = Y^{\mathbf{2}_\mathcal{E}}_{M^\uparrow}$. If $G^q_{M^\uparrow}$ is a prime node, then we set $X^q_* = \pi_{M^\uparrow}(X_* \cap M^\uparrow)$ and $\overline{c}_* = |X_* \cap M^\uparrow|$, $\overline{w}_* = \mathbf{w}(X_* \cap M^\uparrow)$, $\overline{v}_* = |X^q_*|$, $\overline{e}_* = |E(G^q_{M^\uparrow}[X^q_*])|$. Hence, we have that $X_* \cap M^\uparrow \in \mathcal{R}^{\overline{c}_*,\overline{w}_*,\overline{v}_*,\overline{e}_*}_{M^\uparrow}$ and $X_* \cap M^\uparrow \in \mathcal{S}^{\overline{c}_*,\overline{w}_*,\overline{v}_*,\overline{e}_*}_{M^\uparrow}$. By max-isolation of $X_* \cap M^\uparrow$, we therefore have $|\mathcal{S}^{\overline{c}_*,\overline{w}_*,\overline{v}_*,\overline{e}_*}_{M^\uparrow}| = 1$ and Lemma 6.6.20 shows that $|\mathcal{Q}^{\overline{c}_*,\overline{w}_*,\overline{v}_*,\overline{e}_*}_{M^\uparrow}| \not\equiv_{2^{\overline{v}_* - \overline{e}_* + 1}} 0$, so $(\overline{c}_*, \overline{w}_*) \in P_{M^\uparrow}$. Also, $(\overline{c}_*, \overline{w}_*)$ must be the lexicographic maximum in $P_{M^\uparrow}$. Therefore, Definition 6.6.21 must pick $Y^{\mathbf{2}_\mathcal{E}}_{M^\uparrow} = X_* \cap M^\uparrow$; we must have $|X_* \cap M^\uparrow| > |Y^{\mathbf{2}_\mathcal{I}}_{M^\uparrow}|$, since $G[M^\uparrow]$ contains an edge and $X_* \cap M^\uparrow \in \mathcal{F}_{opt}(G[M^\uparrow])$. This concludes the proof of the base case.

Now, when proving the three properties for some $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$, we can inductively assume that they hold for all $M \in \texttt{children}(M^\uparrow)$. The argument for the inductive step is essentially the same as for the base case, however $\mathbf{s}(M) = \mathbf{2}_\mathcal{E}$ can occur now, but for this case we can apply the already proven properties. The first two properties for the child modules allow us to establish $X_* \cap M^\uparrow \in \mathcal{R}_{M^\uparrow}$ even in the inductive step. From that point on, the same argument considering the sets $\mathcal{Q}^{\overline{c},\overline{w},\overline{v},\overline{e}}_{M^\uparrow}$ can be followed to also obtain the first and second property for $M^\uparrow$. □

6.6.5 Inner Dynamic Programming Algorithm

We now need to show how to compute the values $|\mathcal{Q}^{\overline{c},\overline{w},\overline{v},\overline{e}}_{M^\uparrow}|$ modulo $2^{\overline{v}-\overline{e}+1}$ for all $\overline{c}, \overline{w}, \overline{v}, \overline{e}$ when $G^q_{M^\uparrow}$ is prime, from which we can then obtain the $M^\uparrow$-candidate forest $Y^{\mathbf{2}_\mathcal{E}}_{M^\uparrow}$ and proceed through the modular decomposition. We will compute these values by performing dynamic programming along the tree decomposition of the quotient graph $G^q_{M^\uparrow} = G[M^\uparrow]/\texttt{children}(M^\uparrow)$.

Precomputed Data. Let us fix some $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$ and recap the data that is available from solving the previous subproblems. For every $M \in \texttt{children}(M^\uparrow)$, we know the values

- $c^{\mathbf{1}}_M = |Y^{\mathbf{1}}_M| = 1$, $w^{\mathbf{1}}_M = \mathbf{w}(Y^{\mathbf{1}}_M)$,
- $c^{\mathbf{2}_\mathcal{I}}_M = |Y^{\mathbf{2}_\mathcal{I}}_M|$, $w^{\mathbf{2}_\mathcal{I}}_M = \mathbf{w}(Y^{\mathbf{2}_\mathcal{I}}_M)$,
- $c^{\mathbf{2}_\mathcal{E}}_M = |Y^{\mathbf{2}_\mathcal{E}}_M|$, $w^{\mathbf{2}_\mathcal{E}}_M = \mathbf{w}(Y^{\mathbf{2}_\mathcal{E}}_M)$.

The algorithm also knows the sets $Y^{\mathbf{1}}_M$ and $Y^{\mathbf{2}_\mathcal{I}}_M$, but not the sets $Y^{\mathbf{2}_\mathcal{E}}_M$, they will be used in the analysis however. Furthermore, we are given a tree decomposition $(\mathcal{T}^q_{M^\uparrow}, (B^q_t)_{t \in V(\mathcal{T}^q_{M^\uparrow})})$ of the quotient graph $G^q_{M^\uparrow}$ of width k which can be assumed to be very nice by Lemma 2.4.2. To lighten the notation, we do not annotate the bags B^q_t with $M^\uparrow$, but keep in mind that there is a different tree decomposition for each quotient graph.

Definition 6.6.23. Let $t \in V(\mathcal{T}^q_{M^\uparrow})$ be a node of the tree decomposition $\mathcal{T}^q_{M^\uparrow}$. The set of *relaxed solutions* $\mathcal{R}_{t,M^\uparrow}$ consists of the vertex subsets $X \subseteq V_t = \pi^{-1}_{M^\uparrow}(V^q_t)$ that satisfy the following properties:

- X is $M^\uparrow$-forest-nice with respect to G^q_t,
- X has $M^\uparrow$-substructure,
- $\forall M \in \texttt{children}(M^\uparrow)$:
  $\varphi_X(M) = \mathbf{2}_\mathcal{E} \to (M \subseteq V_t \setminus B_t$ or $G[M]$ is a clique of size at least 2),
- $\forall v^q_M \in V^q_t \setminus B^q_t \colon (|X \cap M| \geq 2 \wedge \deg_{G^q_t[\pi_{M^\uparrow}(X)]}(v^q_M) = 0) \to X \cap M = Y^{\mathbf{2}_\mathcal{E}}_M$.

Let $\hat{r}$ be the root node of the tree decomposition $\mathcal{T}^q_{M^\uparrow}$, we want this definition to achieve $\mathcal{R}_{\hat{r},M^\uparrow} = \mathcal{R}_{M^\uparrow}$. Hence, the first two properties are a natural requirement. The third and fourth property lead to the $M^\uparrow$-promotion property at the root node $\hat{r}$ and are more intricate to facilitate the dynamic program. To be precise, since the the bag $B^q_{\hat{r}}$ at the root node $\hat{r}$ is empty, the third property is trivially satisfied and the fourth property turns into the $M^\uparrow$-promotion property.

We exclude the current bag from consideration, because we only want to check whether a module M is isolated in X once all incident edges have been introduced. This is certainly the case when M leaves the current bag, i.e., it is forgotten. If M is isolated at this point, we can safely replace the independent set $Y^{\mathbf{2}_\mathcal{I}}_M$ inside M by the induced forest $Y^{\mathbf{2}_\mathcal{E}}_M$, which cannot decrease the size of X. This means, with the exception of modules inducing a clique, that no module M in the current bag satisfies $\varphi_X(M) = \mathbf{2}_\mathcal{E}$.

The naive dynamic programming routine would not use promotion and track in which modules of the current bag the solution chooses an induced forest (and not just an independent set). By using promotion, we can save this state and only handle the remaining states, namely choosing no vertex, a single vertex, or an independent set. Thereby, we obtain an improved running time.

Due to Lemma 6.6.20, we want to count for each $X \in \mathcal{R}_{t,M^\uparrow}$ the number of consistent homogeneous cuts. Before considering cuts, each module M in the considered bag has four possible states. The intersection with X can be empty, contain a single vertex, or contain at least two vertices, and in the latter case we distinguish whether X intersects a neighboring module or not. To count the homogeneous cuts naively, we would split all states except the empty state into two states, one for each side of a cut, thus obtaining seven total states. However, it turns out that tracking the cut side is not necessary when X intersects M in at least two vertices. When M is isolated, we can simply count it twice, and otherwise M inherits the cut side from the unique neighboring module that is also intersected by X. Hence, five states suffice and we define the cut solutions accordingly.

Definition 6.6.24. Let $t \in V(\mathcal{T}^q_{M^\uparrow})$ be a node of the tree decomposition $\mathcal{T}^q_{M^\uparrow}$. The set of *cut solutions* $\mathcal{Q}_{t,M^\uparrow}$ consists of pairs $(X, (X_L, X_R))$ such that $X \in \mathcal{R}_{t,M^\uparrow}$ and (X_L, X_R) is $M^\uparrow$-homogeneous and a consistent cut of $G_t[X \setminus (\mathrm{iso}_t(X) \cap B_t)]$, where $\mathrm{iso}_t(X) = \bigcup\{M \in \texttt{children}(M^\uparrow) : |X \cap M| \geq 2, \deg_{G^q_t[\pi_{M^\uparrow}(X)]}(v^q_M) = 0\}$.

In the case of isolated modules, we consider it easier to account for the cut side when forgetting the module. Hence, the cuts considered in the definition of $\mathcal{Q}_{t,M^\uparrow}$ do not cover such modules that belong to the current bag B_t. Again, for the root node $\hat{r}$ of the tree

decomposition $\mathcal{T}^q_{M^\uparrow}$, this extra property will be trivially satisfied as the associated bag is empty, hence the definition again achieves that $\mathcal{Q}_{\hat{r},M^\uparrow} = \mathcal{Q}_{M^\uparrow}$

Our dynamic programming algorithm has to track certain additional data of a solution X, namely its size $\overline{c} = |X|$, its weight $\overline{w} = \mathbf{w}(X)$ for the isolation lemma, the number $\overline{v} = |\pi_{M^\uparrow}(X)|$ of intersected modules, and the number $\overline{e} = |E(G^q_t[\pi_{M^\uparrow}(X)])|$ of induced edges in the currently considered subgraph G^q_t of the quotient graph $G^q_{M^\uparrow}$. We need $\overline{v}$ and $\overline{e}$ to apply Lemma 6.6.14. Accordingly, we define $\mathcal{R}^{\overline{c},\overline{w},\overline{v},\overline{e}}_{t,M^\uparrow} = \{X \in \mathcal{R}_{t,M^\uparrow} : \overline{c} = |X \setminus B_t|, \overline{w} = \mathbf{w}(X \setminus B_t), \overline{v} = |\pi_{M^\uparrow}(X) \setminus B^q_t|, \overline{e} = |E(G^q_t[\pi_{M^\uparrow}(X)])|\}$ and $\mathcal{Q}^{\overline{c},\overline{w},\overline{v},\overline{e}}_{t,M^\uparrow} = \{(X,(X_L,X_R)) \in \mathcal{Q}_{t,M^\uparrow} : X \in \mathcal{R}^{\overline{c},\overline{w},\overline{v},\overline{e}}_{t,M^\uparrow}\}$. Note that we exclude the current bag in these counts, except for $\overline{e}$, hence we have to update these counts when we forget a module. This choice simplifies some recurrences in the algorithm, otherwise updating the counts would be a bit cumbersome due to promotion.

Finally, we can define the table that is computed at each node $t \in V(\mathcal{T}^q_{M^\uparrow})$ by our dynamic programming algorithm. Every module M in the current bag has one of five states for a given solution X, these states are denoted by $\mathbf{States} = \{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R, \mathbf{2}_0, \mathbf{2}_1\}$. The bold number refers to the size of the intersection $X \cap M$, i.e., $\mathbf{0}$ if $X \cap M = \emptyset$, $\mathbf{1}$ if $|X \cap M| = 1$, and $\mathbf{2}$ if $|X \cap M| \geq 2$. For $\mathbf{1}$, we additionally track whether the module belongs to the left ($\mathbf{1}_L$) or right side ($\mathbf{1}_R$) of the considered homogeneous cut. For $\mathbf{2}$, we additionally track how many neighboring modules are intersected by X, due to the definition of $M^\uparrow$-forest-nice this number is either zero ($\mathbf{2}_0$) or one ($\mathbf{2}_1$). As argued before, we will not have any modules M with $\varphi_X(M) = \mathbf{2}_\mathcal{E}$ in the current bag unless M induces a clique.

We remark that there is an edge case when the graph $G[M]$ is a clique of size at least 2, as in that case the maximum independent sets of $G[M]$ are simply singletons which are captured by the states $\mathbf{1}_L$ and $\mathbf{1}_R$. As we do not track the degree of such states, we cannot safely perform promotion for them. Instead we directly introduce induced forests inside M in this exceptional case with the state $\mathbf{2}_1$.

Definition 6.6.25. Let $t \in V(\mathcal{T}^q_{M^\uparrow})$ be a node of the tree decomposition $\mathcal{T}^q_{M^\uparrow}$. A function $f\colon B^q_t \to \mathbf{States}$ is called a *t-signature*. Let $(X,(X_L,X_R)) \in \mathcal{Q}_{t,M^\uparrow}$ and $X^q = \pi_{M^\uparrow}(X)$. We say that $(X,(X_L,X_R))$ is *compatible* with a t-signature f if the following properties hold for every $v^q_M \in B^q_t$:

- $f(v^q_M) = \mathbf{0} \to \varphi_X(M) = \mathbf{0}$,
- $f(v^q_M) = \mathbf{1}_L \to \varphi_X(M) = \mathbf{1}$ and $X \cap M \subseteq X_L$,
- $f(v^q_M) = \mathbf{1}_R \to \varphi_X(M) = \mathbf{1}$ and $X \cap M \subseteq X_R$,
- $f(v^q_M) = \mathbf{2}_0 \to \varphi_X(M) = \mathbf{2}_\mathcal{I}$ and $\deg_{G^q_t[X^q]}(v^q_M) = 0$,
- $(f(v^q_M) = \mathbf{2}_1$ and $G[M]$ is not a clique$) \to (\varphi_X(M) = \mathbf{2}_\mathcal{I}$ and $\deg_{G^q_t[X^q]}(v^q_M) = 1)$,
- $(f(v^q_M) = \mathbf{2}_1$ and $G[M]$ is a clique$) \to \varphi_X(M) = \mathbf{2}_\mathcal{E}$.

For a t-signature f, we let $\mathcal{A}_{t,M^\uparrow}(f)$ denote the set of all $(X,(X_L,X_R)) \in \mathcal{Q}_{t,M^\uparrow}$ that are compatible with f. Similarly, we define $\mathcal{A}^{\overline{c},\overline{w},\overline{v},\overline{e}}_{t,M^\uparrow}(f)$ for given $\overline{c} \in [0,\mathbf{c}(M^\uparrow)]$, $\overline{w} \in [0,\mathbf{w}(M^\uparrow)]$, $\overline{v} \in [0,|M^\uparrow|]$, and $\overline{e} \in [0,\overline{v}-1]$.

Fix a parent module $M^\uparrow \in \mathcal{M}^*_{\mathrm{tree}}(G)$ and for every node $t \in V(\mathcal{T}^q_{M^\uparrow})$, t-signature f, and appropriate $\overline{c},\overline{w},\overline{v},\overline{e}$, define the value $A^{\overline{c},\overline{w},\overline{v},\overline{e}}_t(f) = |\mathcal{A}^{\overline{c},\overline{w},\overline{v},\overline{e}}_{t,M^\uparrow}(f)|$. Whenever at least one of $\overline{c},\overline{w},\overline{v},\overline{e}$ is negative, we assume that $A^{\overline{c},\overline{w},\overline{v},\overline{e}}_t(f) = 0$. We now describe the dynamic

programming recurrences to compute $A_t^{\overline{c},\overline{w},\overline{v},\overline{e}}(f)$ for all choices of t, f, $\overline{c},\overline{w},\overline{v},\overline{e}$ based on the type of the node t in the very nice tree decomposition $\mathcal{T}^q_{M\uparrow}$.

Leaf Bag. We have that $V_t^q = B_t^q = \emptyset$ and t has no child. Therefore, the only candidate is $(\emptyset,(\emptyset,\emptyset))$ and we simply need to check if the trackers $\overline{c},\overline{w},\overline{v},\overline{e}$ agree with that:

$$A_t^{\overline{c},\overline{w},\overline{v},\overline{e}}(f) = [\overline{c} = \overline{w} = \overline{e} = \overline{v} = 0]$$

Introduce Vertex Bag. We have that $B_t^q = B_s^q \cup \{v_M^q\}$, where $v_M^q \notin B_s^q$ and s is the only child of t. For the sake of the write-up, we assume that f is an s-signature here. The recurrence is straightforward with the exception of handling the clique case:

$$A_t^{\overline{c},\overline{w},\overline{v},\overline{e}}(f[v_M^q \mapsto \mathbf{s}]) = \begin{cases} A_s^{\overline{c},\overline{w},\overline{v},\overline{e}}(f), & \text{if } \mathbf{s} \in \{\mathbf{0},\mathbf{1}_L,\mathbf{1}_R\}, \\ [G[M] \text{ is not a clique}]A_s^{\overline{c},\overline{w},\overline{v},\overline{e}}(f), & \text{if } \mathbf{s} = \mathbf{2}_0, \\ [|M| > 1 \text{ and } G[M] \text{ is a clique}]A_s^{\overline{c},\overline{w},\overline{v},\overline{e}}(f), & \text{if } \mathbf{s} = \mathbf{2}_1. \end{cases}$$

If $G[M]$ is a clique, then $\varphi_X(M) = \mathbf{2}_{\mathcal{I}}$ can never be satisfied. So, we will directly generate solutions with $\varphi_X(M) = \mathbf{2}_{\mathcal{E}}$ in this case. If $G[M]$ is not a clique, such solutions will only be generated at forget nodes by *promotion*. Recall that no edges incident to M have been introduced yet, which in particular rules out the case that $f(v_M^q) = \mathbf{2}_1$ when $G[M]$ is not a clique, and the trackers are only updated when we forget a module.

Introduce Edge Bag. We have that $\{v_{M_1}^q, v_{M_2}^q\} \subseteq B_t^q = B_s^q$, where $\{v_{M_1}^q, v_{M_2}^q\}$ denotes the introduced edge and s is the only child of t. Define helper functions $\mathrm{edge},\mathrm{feas}\colon \mathbf{States} \times \mathbf{States} \to \{0,1\}$ by $\mathrm{edge}(\mathbf{s}_1,\mathbf{s}_2) = [\mathbf{s}_1 \neq \mathbf{0} \wedge \mathbf{s}_2 \neq \mathbf{0}]$ and feas is given by the following table:

feas	$\mathbf{0}$	$\mathbf{1}_L$	$\mathbf{1}_R$	$\mathbf{2}_0$	$\mathbf{2}_1$
$\mathbf{0}$	1	1	1	1	1
$\mathbf{1}_L$	1	1	0	0	1
$\mathbf{1}_R$	1	0	1	0	1
$\mathbf{2}_0$	1	0	0	0	0
$\mathbf{2}_1$	1	1	1	0	0

The feas-function is used to filter partial solutions that have incompatible states at the newly introduced edge. There are three reasons why states might be incompatible: they belong to different sides of the cut, they directly induce a cycle, or they do not correctly account for the degree in the graph induced by the partial solution.

Furthermore, given a t-signature f, we define the s-signature $\tilde{f}$ as follows. We set $\tilde{f} := f$ if $\mathrm{feas}(f(v_{M_1}^q), f(v_{M_2}^q)) = 0$ or $\mathrm{edge}(f(v_{M_1}^q), f(v_{M_2}^q)) = 0$ or $\mathbf{2}_1 \notin \{f(v_{M_1}^q), f(v_{M_2}^q)\}$. Otherwise, the introduced edge changes the state from $\mathbf{2}_0$ to $\mathbf{2}_1$ at one of its endpoints, i.e., without loss of generality $f(v_{M_1}^q) = \mathbf{2}_1$ and $f(v_{M_2}^q) \in \{\mathbf{1}_L,\mathbf{1}_R\}$ (else, swap role of M_1 and M_2) and we set $\tilde{f} := f[v_{M_1}^q \mapsto \mathbf{2}_0]$. Finally, the recurrence is given by

$$A_t^{\overline{c},\overline{w},\overline{v},\overline{e}}(f) = \mathrm{feas}(f(v_{M_1}^q), f(v_{M_2}^q))A_s^{\overline{c},\overline{w},\overline{v},\overline{e}-\mathrm{edge}(f(v_{M_1}^q),f(v_{M_2}^q))}(\tilde{f}).$$

Observe that we update the edge count, if necessary, in this recurrence. We remark that if $f(v^q_{M_1}) = \mathbf{2}_1$ and $f(v^q_{M_2}) \in \{\mathbf{1}_L, \mathbf{1}_R\}$ and $G[M_1]$ is a clique, we should filter as well, because this means $\varphi_X(M_1) = \mathbf{2}_\mathcal{E}$ and hence $v^q_{M_1}$ should not receive incident edges in $G^q_t[\pi_{M^\uparrow}(X)]$. One could explicitly adapt the recurrence for this case or instead, as we do, observe that since $\varphi_X(M_1) = \mathbf{2}_\mathcal{I}$ is impossible, all entries $A^{\overline{c},\overline{w},\overline{v},\overline{e}}_s(\tilde{f})$ will be zero due to $\tilde{f}(v^q_{M_1}) = \mathbf{2}_0$ and hence we do not generate any partial solutions for this case anyway.

Forget Vertex Bag. We have that $B^q_t = B^q_s \setminus \{v^q_M\}$, where $v^q_M \in B^q_s$ and s is the only child of t. Recall that $c^{\mathbf{2}_\mathcal{I}}_M$, $c^{\mathbf{2}_\mathcal{E}}_M$, $w^{\mathbf{1}}_M$, $w^{\mathbf{2}_\mathcal{I}}_M$, $w^{\mathbf{2}_\mathcal{E}}_M$ denote the size or weight of a singleton set, maximum independent set, or the candidate forest inside M, respectively. The recurrence is given by:

$$
\begin{aligned}
A^{\overline{c},\overline{w},\overline{v},\overline{e}}_t(f) \quad = \quad & A^{\overline{c},\overline{w},\overline{v},\overline{e}}_s(f[v^q_M \mapsto \mathbf{0}]) \\
+ \quad & A^{\overline{c}-1,\overline{w}-w^{\mathbf{1}}_M,\overline{v}-1,\overline{e}}_s(f[v^q_M \mapsto \mathbf{1}_L]) \\
+ \quad & A^{\overline{c}-1,\overline{w}-w^{\mathbf{1}}_M,\overline{v}-1,\overline{e}}_s(f[v^q_M \mapsto \mathbf{1}_R]) \\
+ \quad 2 \quad & A^{\overline{c}-c^{\mathbf{2}_\mathcal{E}}_M,\overline{w}-w^{\mathbf{2}_\mathcal{E}}_M,\overline{v}-1,\overline{e}}_s(f[v^q_M \mapsto \mathbf{2}_0]) \\
+ \quad [G[M] \text{ is not a clique}] \quad & A^{\overline{c}-c^{\mathbf{2}_\mathcal{I}}_M,\overline{w}-w^{\mathbf{2}_\mathcal{I}}_M,\overline{v}-1,\overline{e}}_s(f[v^q_M \mapsto \mathbf{2}_1]) \\
+ \quad 2[|M| > 1 \text{ and } G[M] \text{ is a clique}] \quad & A^{\overline{c}-c^{\mathbf{2}_\mathcal{E}}_M,\overline{w}-w^{\mathbf{2}_\mathcal{E}}_M,\overline{v}-1,\overline{e}}_s(f[v^q_M \mapsto \mathbf{2}_1])
\end{aligned}
$$

As M leaves the current bag, we need to update the trackers $\overline{c}$, $\overline{w}$, and $\overline{v}$. The first three cases are straightforward, but the latter three deserve an explanation. If M had state $\mathbf{2}_0$ before, then $M \subseteq \mathrm{iso}_s(X)$ and $G[M]$ cannot be a clique, so we want to promote the independent set in M to an induced forest and also track the cut side now. Since M remains isolated, both cut sides are possible, explaining the factor 2. If $G[M]$ is not a clique and M had state $\mathbf{2}_1$ before, then we keep the independent set in M and its cut side is already tracked. If instead $G[M]$ is a clique and had state $\mathbf{2}_1$ before, then $M \subseteq \mathrm{iso}_s(X)$ and we are taking an edge (= maximum induced forest) inside M and we need to track its cut side now.

Join Bag. We have that $B^q_t = B^q_{s_1} = B^q_{s_2} = V^q_{s_1} \cap V^q_{s_2}$, where s_1 and s_2 are the two children of t. To state the recurrence for the join bag, we first introduce the *induced forest join* $\oplus_{\mathbf{if}} \colon \mathbf{States} \times \mathbf{States} \to \mathbf{States} \cup \{\bot\}$, where $\bot$ stands for an undefined value, which is defined by the following table:

$\oplus_{\mathbf{if}}$	$\mathbf{0}$	$\mathbf{1}_L$	$\mathbf{1}_R$	$\mathbf{2}_0$	$\mathbf{2}_1$
$\mathbf{0}$	$\mathbf{0}$	$\bot$	$\bot$	$\bot$	$\bot$
$\mathbf{1}_L$	$\bot$	$\mathbf{1}_L$	$\bot$	$\bot$	$\bot$
$\mathbf{1}_R$	$\bot$	$\bot$	$\mathbf{1}_R$	$\bot$	$\bot$
$\mathbf{2}_0$	$\bot$	$\bot$	$\bot$	$\mathbf{2}_0$	$\mathbf{2}_1$
$\mathbf{2}_1$	$\bot$	$\bot$	$\bot$	$\mathbf{2}_1$	$\bot$

When combining two partial solutions, one coming from child s_1 and the other one coming from s_2, we want to ensure that they have essentially the same states on $B^q_t = V^q_{s_1} \cap V^q_{s_2}$. However for the state $\mathbf{2}_1$ (if the considered modules does not induce a clique), we need to decide which child contributes the incident edge in the quotient graph and ensure

that the other child does not contribute an additional edge. This is implemented by the operation $\oplus_{\mathbf{if}}$. Given some set S and functions $f, g\colon S \to \mathbf{States}$, we abuse notation and let $f \oplus_{\mathbf{if}} g\colon S \to \mathbf{States} \cup \{\bot\}$ denote the function obtained from f and g by pointwise application of $\oplus_{\mathbf{if}}$. We also define $\oplus_{\mathbf{2}} = \oplus_{\mathbf{if}}\big|_{\{\mathbf{2}_0,\mathbf{2}_1\}\times\{\mathbf{2}_0,\mathbf{2}_1\}}$ and similarly extend it to functions.

For any module M with $v_M^q \in B_t^q$ that induces a clique, the state $\mathbf{2}_1$ behaves differently and should agree on both children. Hence, we define the following set $\widetilde{B}_t^q = \{v_M^q \in B_t^q : G[M] \text{ is a clique}\}$. We state a first version of the recurrence, which will be transformed further to enable efficient computation:

$$A_t^{\overline{c},\overline{w},\overline{v},\overline{e}}(f) = \sum_{\substack{\overline{c}_1+\overline{c}_2=\overline{c}\\ \overline{w}_1+\overline{w}_2=\overline{w}}} \sum_{\substack{\overline{v}_1+\overline{v}_2=\overline{v}\\ \overline{e}_1+\overline{e}_2=\overline{e}}} \sum_{\substack{f_1,f_2\colon B_t^q\setminus\widetilde{B}_t^q\to\mathbf{States}\colon\\ f_1\oplus_{\mathbf{if}} f_2=f}} A_{s_1}^{\overline{c}_1,\overline{w}_1,\overline{v}_1,\overline{e}_1}(f_1\cup f\big|_{\widetilde{B}_t^q}) A_{s_2}^{\overline{c}_2,\overline{w}_2,\overline{v}_2,\overline{e}_2}(f_2\cup f\big|_{\widetilde{B}_t^q}),$$

where we ensure that all states agree for modules inducing cliques and otherwise apply the induced forest join $\oplus_{\mathbf{if}}$.

To compute this recurrence quickly, we separately handle the part of $\oplus_{\mathbf{if}}$ that essentially checks for equality and reduce the remaining part to already known the results. Given a t-signature $f\colon B_t^q \to \mathbf{States}$, we define $D_t^{=}(f) := \widetilde{B}_t^q \cup f^{-1}(\{\mathbf{0},\mathbf{1}_L,\mathbf{1}_R\})$ and $D_t^{\neq}(f) := B_t^q \setminus D_t^{=}(f)$. We decompose f into $f^{=} := f\big|_{D_t^{=}(f)}$ and $f^{\neq} := f\big|_{D_t^{\neq}(f)}$.

We fix the values $\overline{c},\overline{w},\overline{v},\overline{e}$ and a function $g\colon S \to \mathbf{States}$ where $\widetilde{B}_t^q \subseteq S \subseteq B_t^q$ is some subset of the current bag containing the clique modules. We claim that the entries $A_t^{\overline{c},\overline{w},\overline{v},\overline{e}}(f)$ for all t-signatures f with $f^{=} = g$ (including $D_t^{=}(f) = S$) can be computed in time $\mathcal{O}^*(2^{|B_t^q\setminus S|})$. We branch on $\vec{\mathbf{x}}_1 = (\overline{c}_1,\overline{w}_1,\overline{v}_1,\overline{e}_1)$, which determines the values $\vec{\mathbf{x}}_2 = (\overline{c}_2,\overline{w}_2,\overline{v}_2,\overline{e}_2)$, and define the auxiliary table $T_g^{\vec{\mathbf{x}}_1,\vec{\mathbf{x}}_2}$ indexed by $h\colon B_t^q \setminus S \to \{\mathbf{2}_0,\mathbf{2}_1\}$ as follows

$$T_g^{\vec{\mathbf{x}}_1,\vec{\mathbf{x}}_2}(h) = \sum_{\substack{h_1,h_2\colon B_t^q\setminus S\to\{\mathbf{2}_0,\mathbf{2}_1\}\colon\\ h_1\oplus_{\mathbf{2}} h_2=h}} A_{s_1}^{\overline{c}_1,\overline{w}_1,\overline{v}_1,\overline{e}_1}(g\cup h_1) A_{s_2}^{\overline{c}_2,\overline{w}_2,\overline{v}_2,\overline{e}_2}(g\cup h_2).$$

Since $\oplus_{\mathbf{2}}$ is essentially the same as addition over $\{0,1\}$ with $1+1$ being undefined, we can compute all entries of $T_g^{\vec{\mathbf{x}}_1,\vec{\mathbf{x}}_2}$ in time $\mathcal{O}^*(2^{|B_t^q\setminus S|})$ by the work of, e.g., van Rooij [162, Theorem 2] using fast subset convolution and the fast fourier transform. Then, for every t-signature f with $f^{=} = g$, we obtain $A_t^{\overline{c},\overline{w},\overline{v},\overline{e}}(f)$ by summing $T_g^{\vec{\mathbf{x}}_1,\vec{\mathbf{x}}_2}(f^{\neq})$ over all $\vec{\mathbf{x}}_1 + \vec{\mathbf{x}}_2 = (\overline{c},\overline{w},\overline{v},\overline{e})$. Since there are only polynomially many choices for $\vec{\mathbf{x}}_1$ and $\vec{\mathbf{x}}_2$, this proves the claim.

In conclusion, to compute $A_t^{\overline{c},\overline{w},\overline{v},\overline{e}}(f)$ for all $\overline{c},\overline{w},\overline{v},\overline{e}$, f, we need time

$$\begin{aligned}\sum_{\widetilde{B}_t^q\subseteq S\subseteq B_t^q} \sum_{g\colon S\to\{\mathbf{0},\mathbf{1}_L,\mathbf{1}_R\}} \mathcal{O}^*(2^{|B_t^q\setminus S|}) &\leq \sum_{S\subseteq B_t^q} \mathcal{O}^*(3^{|S|}2^{|B_t^q\setminus S|}) = \mathcal{O}^*\left(\sum_{i=0}^{|B_t^q|}\binom{|B_t^q|}{i}3^i 2^{|B_t^q|-i}\right)\\ &= \mathcal{O}^*((3+2)^{|B_t^q|}) = \mathcal{O}^*(5^k).\end{aligned}$$

Lemma 6.6.26. *Let $M^\uparrow \in \mathcal{M}^*_{\text{tree}}(G)$ be a prime node and $\mathbf{w}\colon V \to [2|V|]$ a weight function. Given a tree decomposition of $G^q_{M^\uparrow}$ of width k and the sets $Y^{\mathbf{1}}_M$, $Y^{\mathbf{2}_\mathcal{I}}_M$ and values $c^{\mathbf{2}_\mathcal{E}}_M$, $w^{\mathbf{2}_\mathcal{E}}_M$ for all $M \in \texttt{children}(M^\uparrow)$, the values $|\mathcal{Q}^{\overline{c},\overline{w},\overline{v},\overline{e}}_{M^\uparrow}|$ can be computed in time $\mathcal{O}^*(5^k)$ for all $\overline{c},\overline{w},\overline{v},\overline{e}$.*

Proof. From the sets $Y^{\mathbf{1}}_M$ and $Y^{\mathbf{2}_\mathcal{I}}_M$, we directly obtain the values $w^{\mathbf{1}}_M$, $c^{\mathbf{2}_\mathcal{I}}_M$, $w^{\mathbf{2}_\mathcal{I}}_M$ for all $M \in \texttt{children}(M^\uparrow)$. We then transform the given tree decomposition into a very nice tree decomposition $(\mathcal{T}^q_{M^\uparrow}, (B^q_t)_{t\in V(\mathcal{T}^q_{M^\uparrow})})$ using Lemma 2.4.2 and run the described dynamic programming algorithm described before to compute the values $A^{\overline{c},\overline{w},\overline{v},\overline{e}}_{\hat{r}}(\emptyset)$, where $\hat{r}$ is the root of $\mathcal{T}^q_{M^\uparrow}$, for all appropriate values of $\overline{c},\overline{w},\overline{v},\overline{e}$. Assuming the correctness of the recurrences, we have that $A^{\overline{c},\overline{w},\overline{v},\overline{e}}_{\hat{r}}(\emptyset) = |\mathcal{A}^{\overline{c},\overline{w},\overline{v},\overline{e}}_{\hat{r}}(\emptyset)| = |\mathcal{Q}^{\overline{c},\overline{w},\overline{v},\overline{e}}_{M^\uparrow}|$ by definition and the degeneration of the conditions at $\hat{r}$.

For the running time, note that for every $t \in V(\mathcal{T}^q_{M^\uparrow})$, there are at most $\mathcal{O}^*(5^k)$ table entries $A^{\overline{c},\overline{w},\overline{v},\overline{e}}_t(f)$ and the recurrences can be computed in polynomial time except for the case of join bags. In the case of a join bag, all table entries can be computed simultaneously in time $\mathcal{O}^*(5^k)$. By Lemma 2.4.2 the tree decomposition $\mathcal{T}^q_{M^\uparrow}$ has a polynomial number of nodes, hence the running time follows and it remains to sketch the correctness of the dynamic programming recurrences.

For leaf bags, the correctness follows by observing that $\mathcal{A}_t(\emptyset) = \mathcal{Q}_{t,M^\uparrow} = \{(\emptyset,(\emptyset,\emptyset))\}$. So, we start by considering introduce vertex bags. We set up a bijection between $\mathcal{A}_t(f[v^q_M \mapsto \mathbf{s}])$ and $\mathcal{A}_s(f)$ depending on $\mathbf{s} \in \mathbf{States}$. We map $(X,(X_L,X_R)) \in \mathcal{A}_s(f)$ to

- $(X,(X_L,X_R))$ if $\mathbf{s} = \mathbf{0}$,
- $(X \cup Y^{\mathbf{1}}_M, (X_L \cup Y^{\mathbf{1}}_M, X_R))$ if $\mathbf{s} = \mathbf{1}_L$ ($\mathbf{1}_R$ is analogous),
- $(X \cup Y^{\mathbf{2}_\mathcal{I}}_M, (X_L,X_R))$ if $\mathbf{s} = \mathbf{2}_0$ and $G[M]$ is not a clique,
- $(X \cup Y^{\mathbf{2}_\mathcal{E}}_M, (X_L,X_R))$ if $\mathbf{s} = \mathbf{2}_1$ and $G[M]$ is a clique of size at least 2.

In the last two cases, we have $M \subseteq \text{iso}_t(X)$, so we do not need to track the cut side. Using $M^\uparrow$-substructure it is possible to verify that these mappings constitute bijections. The case that $\mathbf{s} = \mathbf{2}_1$ and $G[M]$ is not a clique is impossible, since no edges incident to v^q_M are introduced yet. The case that $\mathbf{s} = \mathbf{2}_0$ and $G[M]$ is a clique is impossible, since any subset of M of size at least two has to induce an edge.

For introduce edge bags, we highlight the case that $\tilde{f}(v^q_{M_1}) = \mathbf{2}_0$ and $f(v^q_{M_1}) = \mathbf{2}_1$, where M_1 needs to inherit the cut side from M_2. Formally, a partial solution $(X,(X_L,X_R)) \in A^{\overline{c},\overline{w},\overline{v},\overline{e}-1}_s(\tilde{f})$ with $f(v^q_{M_2}) = \mathbf{1}_L$ is bijectively mapped to $(X,(X_L \cup (X \cap M), X_R)) \in A^{\overline{c},\overline{w},\overline{v},\overline{e}}_t(f)$ and analogously when $f(v^q_{M_2}) = \mathbf{1}_R$. We have already argued the correct handling of the clique case when presenting the recurrence. The remaining cases are straightforward.

We proceed with forget vertex bags. First, we observe that all considered cases are disjoint, hence no overcounting occurs. The handling of the cases $\mathbf{0}$, $\mathbf{1}_L$, and $\mathbf{1}_R$ is standard and we omit further explanation. For isolated modules, we need to track the cut side when we forget them, since both sides are possible, we multiply with the factor 2. Furthermore, we need to perform the promotion when we forget a module with state $\mathbf{2}_0$. The most involved case is $Y^{\mathbf{2}_\mathcal{E}}_M \neq Y^{\mathbf{2}_\mathcal{I}}_M$ and $G[M]$ is not a clique, then we perform promotion on the isolated module M, swapping $Y^{\mathbf{2}_\mathcal{I}}_M$ with $Y^{\mathbf{2}_\mathcal{E}}_M$, and now have to track the cut side of M, again yielding the factor 2. Formally, if f is a t-signature and $(X,(X_L,X_R)) \in \mathcal{A}_s(f[v^q_M \mapsto \mathbf{2}_0])$, then $G[M]$

is not a clique and we obtain the solutions $((X \setminus M) \cup Y_M^{\mathbf{2}\varepsilon}, (X_L \cup Y_M^{\mathbf{2}\varepsilon}, X_R)) \in \mathcal{A}_t(f)$ and $((X \setminus M) \cup Y_M^{\mathbf{2}\varepsilon}, (X_L, X_R \cup Y_M^{\mathbf{2}\varepsilon})) \in \mathcal{A}_t(f)$.

For the join bags, we have that $V_{s_1}^q \cap V_{s_2}^q = B_t^q$, so the behavior on the intersection is completely described by the signature f. Every $(X, (X_L, X_R)) \in \mathcal{A}_t(f)$ splits into a solution $(X^1, (X_L^1, X_L^2)) \in \mathcal{A}_{s_1}(f_1)$ at s_1 and a solution $(X^2, (X_L^2, X_R^2)) \in \mathcal{A}_{s_2}(f_2)$ at s_2, where for $i \in [2]$ we set $X^i = X \cap V_{s_i}$, $X_L^i = (X_L \cap V_{s_i}) \setminus (\mathrm{iso}_{s_i}(X^i) \cap B_{s_i})$, $X_R^i = (X_R \cap V_{s_i}) \setminus (\mathrm{iso}_{s_i}(X^i) \cap B_{s_i})$ and

$$f_i(v_M^q) = \begin{cases} f(v_M^q), & \text{if } f(v_M^q) \neq \mathbf{2}_1, \\ \mathbf{2}_1, & \text{if } f(v_M^q) = \mathbf{2}_1 \wedge G[M] \text{ is clique of size} \geq 2, \\ \mathbf{2}_d, & \text{if } f(v_M^q) = \mathbf{2}_1 \wedge G[M] \text{ is not a clique} \wedge \deg_{G_{s_i}^q[\pi_{M^\uparrow}(X^i)]}(v_M^q) = d. \end{cases}$$

For a non-clique module with state $\mathbf{2}_1$, the edge leading to degree 1 is present at one of the child nodes s_1 or s_2, but not at the other one. At the child, where the edge is not present, the module has state $\mathbf{2}_0$ and is isolated, therefore we do not track the cut side and hence have to account for this in the definitions of X_L^i and X_R^i. This map can be seen to be a bijection between $\mathcal{A}_t(f)$ and $\bigcup_{f_1,f_2} \mathcal{A}_{s_1}(f_1) \times \mathcal{A}_{s_2}(f_2)$, where the union is over all $f_1, f_2 \colon B_t^q \to \mathbf{States}$ such that $f\big|_{\widetilde{B}_t^q} = f_1\big|_{\widetilde{B}_t^q} = f_2\big|_{\widetilde{B}_t^q}$ and $f\big|_{B_t^q \setminus \widetilde{B}_t^q} = f_1\big|_{B_t^q \setminus \widetilde{B}_t^q} \oplus_{\mathbf{if}} f_2\big|_{B_t^q \setminus \widetilde{B}_t^q}$, which is implemented by the join-recurrence once we account for the trackers $\overline{c}$, $\overline{w}$, $\overline{v}$, and $\overline{e}$; as every edge is introduced exactly once and the other trackers are only computed for forgotten modules, no overcounting happens here and we only have to distribute the trackers between s_1 and s_2. We remark that the correctness here requires that the promotion property is only applied to forgotten modules which have received all incident edges already. □

Finally, we have assembled all ingredients for the FEEDBACK VERTEX SET algorithm; the only remaining step is to combine them all.

Theorem 6.6.27. *There exists a Monte-Carlo algorithm that, given a tree decomposition of width k for every prime quotient graph in the modular decomposition of G, solves* FEEDBACK VERTEX SET *in time $\mathcal{O}^*(5^k)$. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.*

Proof. Solving the complementary problem INDUCED FOREST, we begin by computing the sets $Y_{M^\uparrow}^{\mathbf{1}}$ and $Y_{M^\uparrow}^{\mathbf{2}\mathcal{I}}$ for all $M^\uparrow \in \mathcal{M}_{\mathrm{tree}}(G)$ in time $\mathcal{O}^*(2^k)$ using Theorem 6.2.3. We sample a weight function $\mathbf{w} \colon V \to [2n]$ uniformly at random, which max-isolates $\mathcal{F}_{opt}(G)$ with probability at least $1/2$ by Lemma 6.6.9. We generate the sets $Y_{M^\uparrow}^{\mathbf{2}\varepsilon}$ for the base cases $M^\uparrow = \{v\}$, $v \in V$.

By bottom-up dynamic programming along the modular decomposition, we inductively compute the values $c_{M^\uparrow}^{\mathbf{2}\varepsilon}$ and $w_{M^\uparrow}^{\mathbf{2}\varepsilon}$, $M^\uparrow \in \mathcal{M}_{\mathrm{tree}}^*(G)$, given the values $c_M^{\mathbf{2}\varepsilon}$ and $w_M^{\mathbf{2}\varepsilon}$ for all $M \in \texttt{children}(M^\uparrow)$. To do so, we distinguish whether $M^\uparrow$ is a parallel, series, or prime node. In the first two cases, we can compute these values in polynomial time by Section 6.6.4.

In the prime case, we can compute the values $|\mathcal{Q}_{M^\uparrow}^{\overline{c},\overline{w},\overline{v},\overline{e}}|$ in time $\mathcal{O}^*(5^k)$ by using Lemma 6.6.26. From these values, we can obtain the values $c_{M^\uparrow}^{\mathbf{2}\varepsilon}$ and $w_{M^\uparrow}^{\mathbf{2}\varepsilon}$ by the description in Section 6.6.4 in polynomial time. As the modular decomposition has a polynomial number of nodes, the running time follows.

If $c_V^{\mathbf{2}^\varepsilon} \geq \bar{b}$, then the algorithm returns true and otherwise the algorithm returns false. It remains to prove the correctness of this step, assuming that the weight function $\mathbf{w}$ is isolating. By Lemma 6.6.22, we have that $Y_V^{\mathbf{2}^\varepsilon}$ is a maximum induced forest of $G[V] = G$ if $\mathbf{w}$ is isolating and since $c_V^{\mathbf{2}^\varepsilon} = |Y_V^{\mathbf{2}^\varepsilon}|$ this shows that the algorithm is correct in this case. Since we always ensure that $Y_V^{\mathbf{2}^\varepsilon}$ is an induced forest, but not necessarily maximum, even if $\mathbf{w}$ is not isolating, the algorithm cannot return false positives. □

7 Lower Bounds Parameterized By Clique-Width

7.1 General Approach

In this section, we prove the tight lower bounds for Connected Vertex Cover and Connected Dominating Set parameterized by linear-clique-width. The high-level construction principle follows the style of Lokshtanov et al. [126] as discussed before. For linear-clique-width, the considered separators of size one are simply *joins*, i.e., all edges between the two sides of the separator are present, each side representing a label of the clique-expression. So, adjacent path gadgets are connected via a join and we study the state behavior across a join to determine appropriate transition orders.

Determining a Transition Order. Due to the joins, we will have to study the *label states* that can be taken by a label with respect to a partial solution of the target problem. As the number of label states can be exponential in the number of vertex states, determining a transition order ad hoc quickly becomes infeasible. Hence, the generic approach of first finding a large triangular submatrix of an appropriate compatibility matrix, which allows us to deduce a transition order, will serve us very well here.

Anatomy of a Path Gadget. Our path gadget design follows a quite generic approach that allows us to easily implement the path gadget states deduced from the triangular submatrix and simple communication with the decoding gadgets. The path gadgets consist of three parts: a *central clique* that communicates with the decoding gadgets, and two *boundary* parts, i.e., the *left* and *right* parts that connect to the previous and following join, respectively. In the central clique, each solution will avoid exactly one vertex representing the state of the path gadget. To implement the transition order, the left and right parts have to communicate appropriate states to the two adjacent path gadgets. The appropriate states are deduced by *pairing* the states along the main diagonal of the triangular submatrix.

Designing the Boundary of a Path Gadget. The state of a vertex in a partial solution consists of several properties: whether it is contained in the partial solution, a connectivity property, and possibly whether it is dominated (for Connected Dominating Set). Our strategy for designing the boundary parts is to *isolate* these properties for the vertices incident to the joins and represent by pairs of *indicator vertices* whether each property is satisfied or not. We ensure that a solution can only pick one vertex per indicator pair. This allows us to generically connect the clique vertices to appropriate indicator vertices, which

then forces the desired states at the boundaries depending on which clique vertex is avoided. This concludes the description of the high-level ideas for constructing the path gadgets.

Decoding and Clause Gadgets. By designing the path gadgets with the central clique as an interface for the decoding gadgets, we can reuse already-known constructions for the decoding and clause gadgets. For the decoding gadget, we use the construction of the CONNECTED VERTEX COVER[pathwidth] lower bound by Cygan et al. [50]. The clause gadget only consists of a vertex that is forced into every solution by a simple problem-dependent construction.

Root-Connectivity. To capture the connectivity constraint of CONNECTED VERTEX COVER and CONNECTED DOMINATING SET, we reuse a construction present in the lower bound proofs of Cygan et al. [50] for connectivity problems parameterized by pathwidth. We create a distinguished vertex $\hat{r}$ called the *root* and by attaching a vertex of degree 1 to $\hat{r}$ we ensure that every connected vertex cover or connected dominating set has to contain $\hat{r}$. Given a vertex subset $X \subseteq V(G)$ with $\hat{r} \in X$, we say that a vertex $v \in X$ is *root-connected* in X if there is a $v, \hat{r}$-path in $G[X]$. We will just say *root-connected* if X is clear from the context. The graph $G[X]$ is connected if and only if all vertices of X are root-connected in X. For the state of a partial solution X, it is important to consider which vertices are root-connected in X and which are not.

7.2 Connected Vertex Cover Lower Bound

This subsection is devoted to proving that CONNECTED VERTEX COVER (with unit costs) cannot be solved in time $\mathcal{O}^*((6-\varepsilon)^{\text{lin-cw}(G)})$ for some $\varepsilon > 0$ unless the CNF-SETH, Conjecture 2.1.1, fails. We first design the path gadget, approaching the design as discussed, and analyze it in isolation and afterwards we present the complete construction and correctness proofs. Afterwards we present the complete construction, where the decoding gadgets are directly adapted from the lower bound for CONNECTED VERTEX COVER[pathwidth] given by Cygan et al. [50].

7.2.1 Path Gadget

To rule out the running time $\mathcal{O}^*((6-\varepsilon)^{\text{lin-cw}(G)})$ for any $\varepsilon > 0$, we build a path gadget that admits 6 distinct states and narrows down to a single label, so that each row of the construction contributes one unit of linear-clique-width. As discussed previously, we begin by analyzing the behavior of a partial solution on a label.

Every single vertex v has one of 3 states with respect to a partial solution X: $v \notin X$ (state $\mathbf{0}$), $v \in X$ and v is root-connected (state $\mathbf{1}_1$) or not (state $\mathbf{1}_0$). First, we observe that it is irrelevant how often each vertex state appears inside a label, rather we only care whether a vertex state appears in a label or not. Hence, we can describe the state of a label as a subset of $\{\mathbf{0}, \mathbf{1}_0, \mathbf{1}_1\}$, where the empty subset is excluded, as we do not consider empty labels.

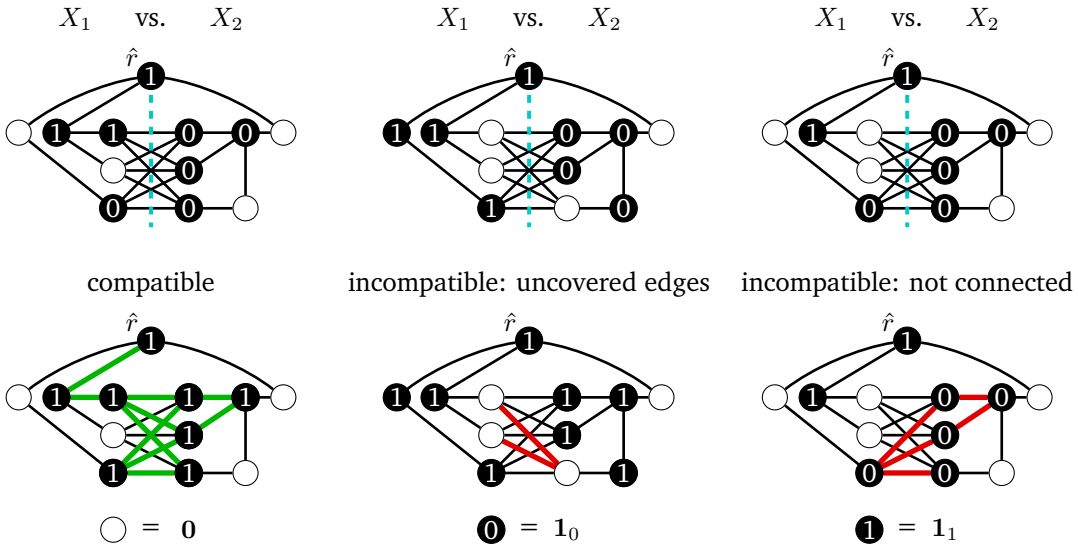


Fig. 7.1.: Several cases of partial solution compatibility across a join. The first row depicts the vertex states in X_1 and X_2, separated by the dashed line. The second row depicts the vertex states in $X_1 \cup X_2$ and highlights, from left to right, the induced edges, the uncovered edges, and a connected component not containing the root $\hat{r}$.

X_1 vs. X_2	$\{\mathbf{0}\}$	$\{\mathbf{1}_0\}$	$\{\mathbf{1}_1\}$	$\{\mathbf{1}_0, \mathbf{0}\}$	$\{\mathbf{1}_1, \mathbf{0}\}$	$\{\mathbf{1}_1, \mathbf{1}_0\}$	$\{\mathbf{1}_1, \mathbf{1}_0, \mathbf{0}\}$
$\{\mathbf{0}\}$	0	0	1	0	0	0	0
$\{\mathbf{1}_0\}$	0	0	1	0	1	1	1
$\{\mathbf{1}_1\}$	1	1	1	1	1	1	1
$\{\mathbf{1}_0, \mathbf{0}\}$	0	0	1	0	0	1	0
$\{\mathbf{1}_1, \mathbf{0}\}$	0	1	1	0	0	1	0
$\{\mathbf{1}_1, \mathbf{1}_0\}$	0	1	1	1	1	1	1
$\{\mathbf{1}_1, \mathbf{1}_0, \mathbf{0}\}$	0	1	1	0	0	1	0

Tab. 7.1.: The compatibility matrix for Connected Vertex Cover. The rows describe the label state of the partial solution X_1 and the columns the label state of X_2. Ones correspond to compatible pairs of label states and zeroes to incompatible pairs.

We proceed by studying the compatibility of these label states across a join, but we will only give an informal description here. Essentially, we assume that the considered join is the final opportunity for two partial solutions $X_1, X_2 \subseteq V$ with $\hat{r} \in X_i$, $i \in [2]$, living on separate sides of the join (with the exception of $\hat{r}$) to connect. Hence, the partial solutions X_1 and X_2 are considered to be *compatible* when in $X_1 \cup X_2$ every vertex incident to the considered join has state $\mathbf{0}$ or $\mathbf{1}_1$ and every edge of the join is covered by $X_1 \cup X_2$; see Figure 7.1. If X_1 and X_2 are compatible, then $X_1 \cup X_2$ *can* be a *global* solution, if outside of the considered join all constraints are satisfied. Since the interaction of X_i, $i \in [2]$, with the respective side of the join is captured by the aforementioned states, we obtain the compatibility matrix in Table 7.1.

To determine a transition order and which states should be implemented by a path gadget, we find the triangular submatrix in Table 7.2, after reordering rows and columns. Note that the triangular submatrix only involves states consisting of at most two vertex states, hence labels consisting of two vertices should be sufficient to generate these states. Indeed, in the forthcoming construction, the labels incident to the join are independent sets of size two and the state sets will be represented by the following ordered pairs of

X_1 vs. X_2	$\{\mathbf{0}\}$	$\{\mathbf{1}_0, \mathbf{0}\}$	$\{\mathbf{1}_0\}$	$\{\mathbf{1}_1, \mathbf{0}\}$	$\{\mathbf{1}_1, \mathbf{1}_0\}$	$\{\mathbf{1}_1\}$
$\{\mathbf{1}_1\}$	1	1	1	1	1	1
$\{\mathbf{1}_1, \mathbf{1}_0\}$	0	1	1	1	1	1
$\{\mathbf{1}_1, \mathbf{0}\}$	0	0	1	0	1	1
$\{\mathbf{1}_0\}$	0	0	0	1	1	1
$\{\mathbf{1}_0, \mathbf{0}\}$	0	0	0	0	1	1
$\{\mathbf{0}\}$	0	0	0	0	0	1

Tab. 7.2.: A large triangular submatrix in the compatibility matrix of CONNECTED VERTEX COVER. The rows and columns have been reordered.

vertex states: $(\mathbf{0}, \mathbf{0})$, $(\mathbf{1}_0, \mathbf{0})$, $(\mathbf{1}_0, \mathbf{1}_0)$, $(\mathbf{1}_1, \mathbf{0})$, $(\mathbf{1}_1, \mathbf{1}_0)$, $(\mathbf{1}_1, \mathbf{1}_1)$. Pairing the states along the diagonal then tells us for each case which states the path gadget should communicate to the left and right boundary respectively. For example, for the third position of the diagonal, we pair $(\mathbf{1}_0, \mathbf{1}_0)$ with $(\mathbf{1}_1, \mathbf{0})$, meaning that for the third state of the transition order, the path gadget should communicate the states $(\mathbf{1}_0, \mathbf{1}_0)$ to the left boundary and the states $(\mathbf{1}_1, \mathbf{0})$ to the right boundary.

Formal Definition of States. We define the three atomic states $\mathbf{Atoms} = \{\mathbf{0}, \mathbf{1}_0, \mathbf{1}_1\}$ and define the two predicates $\mathtt{sol}, \mathtt{conn}\colon \mathbf{Atoms} \to \{0, 1\}$ by $\mathtt{sol}(\mathbf{a}) = [\mathbf{a} \in \{\mathbf{1}_0, \mathbf{1}_1\}]$ and $\mathtt{conn}(\mathbf{a}) = [\mathbf{a} = \mathbf{1}_1]$. The atom $\mathbf{0}$ means that a vertex is not inside the partial solution; $\mathbf{1}_1$ and $\mathbf{1}_0$ indicate that a vertex is inside the partial solution and the subscript indicates whether it is root-connected or not. Building on these atomic states, we define six (gadget) states consisting of four atomic states each:

$$\begin{aligned}
\mathbf{s}^1 &= (\mathbf{0}\ , \mathbf{0}\ , \mathbf{1}_1, \mathbf{1}_1),\\
\mathbf{s}^2 &= (\mathbf{1}_0, \mathbf{0}\ , \mathbf{1}_1, \mathbf{1}_0),\\
\mathbf{s}^3 &= (\mathbf{1}_0, \mathbf{1}_0, \mathbf{1}_1, \mathbf{0}\ ),\\
\mathbf{s}^4 &= (\mathbf{1}_1, \mathbf{0}\ , \mathbf{1}_0, \mathbf{1}_0),\\
\mathbf{s}^5 &= (\mathbf{1}_1, \mathbf{1}_0, \mathbf{1}_0, \mathbf{0}\ ),\\
\mathbf{s}^6 &= (\mathbf{1}_1, \mathbf{1}_1, \mathbf{0}\ , \mathbf{0}\ ).
\end{aligned}$$

The gadget states are numbered in the transition order. We collect the six gadget states in the set $\mathbf{States} = \{\mathbf{s}^1, \ldots, \mathbf{s}^6\}$ and use the notation $\mathbf{s}^\ell_i \in \mathbf{Atoms}$, $i \in [4]$, $\ell \in [6]$, to refer to the i-th atomic component of state $\mathbf{s}^\ell$. Observe that $\mathbf{s}^\ell$ can be obtained from $\mathbf{s}^{7-\ell}$ by swapping the first and second components with the third and fourth components, i.e., $\mathbf{s}^\ell_1 = \mathbf{s}^{7-\ell}_3$ and $\mathbf{s}^\ell_2 = \mathbf{s}^{7-\ell}_4$ for all $\ell \in [6]$.

Given a partial solution $Y \subseteq V(G)$, we formally associate to each vertex its state in Y with the map $\mathbf{state}_Y\colon V(G) \setminus \{\hat{r}\} \to \mathbf{Atoms}$, which is defined by

$$\mathbf{state}_Y(v) = \begin{cases} \mathbf{0} & \text{if } v \notin Y,\\ \mathbf{1}_0 & \text{if } v \in Y \text{ and } v \text{ is not root-connected in } Y \cup \{\hat{r}\},\\ \mathbf{1}_1 & \text{if } v \in Y \text{ and } v \text{ is root-connected in } Y \cup \{\hat{r}\}. \end{cases}$$

Formal Construction. We proceed by describing how to construct the path gadget P. We create the following sets of vertices:

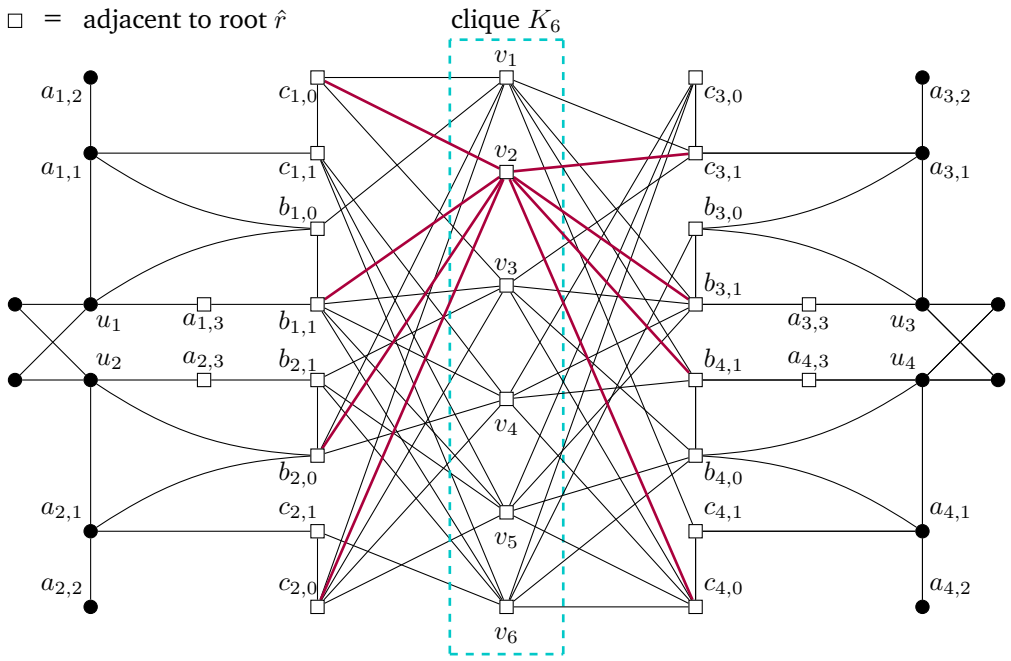


Fig. 7.2.: The path gadget P with the join vertices u_1, u_2 and u_3, u_4 already joined to further vertices. All vertices that are depicted with a rectangle are adjacent to the root vertex $\hat{r}$. The vertices inside the cyan dashed rectangle induce a clique. For sake of understanding, we have highlighted the edges incident to v_2 as one example.

- 4 *join* vertices $u_1, \ldots, u_4$,
- 12 *auxiliary* vertices $a_{1,1}, a_{1,2}, a_{1,3}, a_{2,1}, \ldots, a_{4,3}$,
- 8 *solution indicator* vertices $b_{1,0}, b_{1,1}, b_{2,0}, b_{2,1}, \ldots, b_{4,1}$,
- 8 *connectivity indicator* vertices $c_{1,0}, c_{1,1}, c_{2,0}, c_{2,1}, \ldots, c_{4,1}$, and
- 6 *clique* vertices $v_1, \ldots, v_6$.

For every $i \in [4]$, we add the edges $\{u_i, a_{i,1}\}$, $\{u_i, a_{i,3}\}$, $\{u_i, b_{i,0}\}$, $\{a_{i,1}, a_{i,2}\}$, $\{a_{i,1}, b_{i,0}\}$, $\{a_{i,1}, c_{i,1}\}$, $\{a_{i,3}, b_{i,1}\}$, and $\{b_{i,0}, b_{i,1}\}$ and $\{c_{i,0}, c_{i,1}\}$. Furthermore, we make $a_{i,3}$ for all $i \in [4]$, all solution indicator vertices, all connectivity indicator vertices, and all clique vertices adjacent to the root vertex $\hat{r}$. We add all possible edges between the clique vertices v_ℓ, $\ell \in [6]$, so that they induce a clique of size 6.

Finally, we explain how to connect the indicator vertices to the clique vertices. The clique vertex v_ℓ corresponds to choosing state s^ℓ on the join vertices (u_1, u_2, u_3, u_4). The desired behavior of P is that a partial solution X of $P + \hat{r}$ contains $b_{i,1}$ if and only if X contains u_i and for the connectivity indicators, that X contains $c_{i,1}$ if and only if X contains u_i and u_i is root-connected in X. Accordingly, for all $i \in [4]$ and $\ell \in [6]$, we add the edges $\{v_\ell, b_{i,\mathtt{sol}(\mathsf{s}_i^\ell)}\}$ and $\{v_\ell, c_{i,\mathtt{conn}(\mathsf{s}_i^\ell)}\}$. This concludes the construction of P, see Figure 7.2.

Behavior of a Single Path Gadget. We can now begin with the analysis of the path gadget. To do so, we assume that G is a graph that contains $P + \hat{r}$ as an induced subgraph and that only the join vertices $u_i, i \in [4]$, and clique vertices $v_\ell, \ell \in [6]$, have neighbors outside this copy of $P + \hat{r}$. Furthermore, let X be a connected vertex cover of G with $\hat{r} \in X$. We study the behavior of such connected vertex covers on P; we will abuse notation and write

$X \cap P$ instead of $X \cap V(P)$. The assumption on how P connects to the remaining graph implies that any vertex $v \in V(P)$ with $\mathbf{state}_{X \cap P}(v) = \mathbf{1}_0$ has to be root-connected in X through some path that leaves $P + \hat{r}$ via one of the join vertices u_i, $i \in [4]$. Note that any path leaving $P + \hat{r}$ through some clique vertex v_ℓ, $\ell \in [6]$, immediately yields a path to $\hat{r}$ in $P + \hat{r}$ as $\{v_\ell : \ell \in [6]\} \subseteq N(\hat{r})$.

We obtain a lower bound for $|X \cap P|$ via a vertex-disjoint packing of subgraphs.

Lemma 7.2.1. *We have $|X \cap P| \geq 21 = 4 \cdot 4 + 5$ and $|X \cap \{v_1, \ldots, v_6\}| \geq 5$ and for all $i \in [4]$ that $a_{i,1} \in X$, $|X \cap \{u_i, a_{i,3}, b_{i,1}, b_{i,0}\}| \geq 2$, $|X \cap \{c_{i,0}, c_{i,1}\}| \geq 1$.*

Proof. For all $i \in [4]$, the vertex $a_{i,2}$ has degree 1 in G with unique neighbor $a_{i,1}$, hence we must have $a_{i,1} \in X$ to either connect $a_{i,2}$ to $\hat{r}$ or to cover the edge $\{a_{i,1}, a_{i,2}\}$ if $a_{i,2} \notin X$. The vertices u_i, $a_{i,3}$, $b_{i,1}$, $b_{i,0}$ induce a cycle of length 4 for all $i \in [4]$ and any vertex cover has to contain at least 2 vertices of every such cycle. The edge $\{c_{i,0}, c_{i,1}\}$ has to be covered for all $i \in [4]$. Finally, the clique formed by the vertices $v_1, \ldots, v_6$ has size 6 and any vertex cover has to contain at least 5 vertices of such a clique. Since we only considered pairwise disjoint sets of vertices, these lower bounds simply add up and we obtain $|X \cap P| \geq 4(1 + 2 + 1) + 5 = 21$. □

Using Lemma 7.2.1, we precisely analyze the solutions matching the lower bound of 21 on P and show that such solutions have the desired state behavior. We define for any $Y \subseteq V(P)$ the 4-tuple

$$\mathbf{state}(Y) = (\mathbf{state}_Y(u_1), \mathbf{state}_Y(u_2), \mathbf{state}_Y(u_3), \mathbf{state}_Y(u_4))$$

and the following lemma shows that the states communicated to the boundary depend on the state of the central clique as desired.

Lemma 7.2.2. *If $|X \cap P| \leq 21$, then $|X \cap P| = 21$ and $a_{i,1} \in X$, $X \cap \{u_i, a_{i,3}, b_{i,1}, b_{i,0}\} \in \{\{u_i, b_{i,1}\}, \{a_{i,3}, b_{i,0}\}\}$, $|X \cap \{c_{i,0}, c_{i,1}\}| = 1$ for all $i \in [4]$ and $|X \cap \{v_1, \ldots, v_6\}| = 5$. Furthermore, we have $\mathbf{state}(X \cap P) = \mathbf{s}^\ell$ for the unique integer $\ell \in [6]$ with $v_\ell \notin X$.*

Proof. Due to $|X \cap P| \leq 21$, all the inequalities of Lemma 7.2.1 have to be tight. Since $|X \cap \{u_i, a_{i,3}, b_{i,1}, b_{i,0}\}| = 2$ and $\{u_i, a_{i,3}, b_{i,1}, b_{i,0}\}$ induces a cycle of length 4, X has to contain a pair of antipodal vertices of this cycle, so either the pair $\{u_i, b_{i,1}\}$ or the pair $\{a_{i,3}, b_{i,0}\}$. It remains to prove the last property regarding the states of the join vertices.

Consider any $i \in [4]$, we will show that $\mathbf{state}_{X \cap P}(u_i) = \mathbf{s}_i^\ell$. Since X is a vertex cover and $v_\ell \notin X$, we must have that $N(v_\ell) \subseteq X$. By construction, this means that X must in particular contain the vertices $b_{i,\mathtt{sol}(\mathbf{s}_i^\ell)}$ and $c_{i,\mathtt{conn}(\mathbf{s}_i^\ell)}$ and due to the previous equations X cannot contain the other vertex of each of these indicator pairs. Due to the choice of an antipodal pair we have $u_i \in X$ if and only if $b_{i,1} \in X$ if and only if $\mathtt{sol}(\mathbf{s}_i^\ell) = 1$ if and only if $\mathbf{s}_i^\ell \in \{\mathbf{1}_0, \mathbf{1}_1\}$. If $u_i \in X$, then $a_{i,3} \notin X$ and $b_{i,0} \notin X$, so u_i is root-connected in $X \cap (P + \hat{r})$ if and only if $c_{i,1} \in X$, i.e., via the path $u_i, a_{i,1}, c_{i,1}, \hat{r}$. Furthermore, $c_{i,1} \in X \Leftrightarrow \mathtt{conn}(\mathbf{s}_i^\ell) = 1 \Leftrightarrow \mathbf{s}_i^\ell = \mathbf{1}_1$. This shows that $\mathbf{state}_{X \cap P}(u_i) = \mathbf{s}_i^\ell$ and since this argument applies to all $i \in [4]$, we obtain that $\mathbf{state}(X \cap Y) = \mathbf{s}^\ell$. □

Moving on, we establish that for every state $\mathbf{s}^\ell \in \mathbf{States}$ a partial solution attaining this state actually exists, which we use to prove that every satisfying gives rise to a sufficiently

small connected vertex cover of the constructed graph. Furthermore, for these partial solutions, it is sufficient to check for root-connectivity at the join vertices.

Lemma 7.2.3. *For every $\ell \in [6]$, there exists a vertex cover X_P^ℓ of P such that $|X_P^\ell| = 21$, $X_P^\ell \cap \{v_1, \ldots, v_6\} = \{v_1, \ldots, v_6\} \setminus \{v_\ell\}$, and $\mathbf{state}(X_P^\ell) = \mathbf{s}^\ell$. If X is a vertex cover of G with $\hat{r} \in X$ and $X \cap P = X_P^\ell$ and for every $i \in [4]$ either $u_i \notin X$ or u_i is root-connected in X, then every vertex of X_P^ℓ is root-connected in X.*

Proof. We define

$$\begin{aligned} X_P^\ell = &(\{v_1, \ldots, v_6\} \setminus \{v_\ell\}) \cup \{a_{i,1} : i \in [4]\} \\ &\cup \{u_i : i \in [4] \text{ and } \mathtt{sol}(\mathbf{s}_i^\ell) = 1\} \cup \{a_{i,3} : i \in [4] \text{ and } \mathtt{sol}(\mathbf{s}_i^\ell) = 0\} \\ &\cup \{b_{i,\mathtt{sol}(\mathbf{s}_i^\ell)} : i \in [4]\} \cup \{c_{i,\mathtt{conn}(\mathbf{s}_i^\ell)} : i \in [4]\} \end{aligned}$$

and claim that X_P^ℓ is a vertex cover with the desired properties. We clearly have that $|X_P^\ell| = 21$ and $v_\ell \notin X_P^\ell$. We proceed by showing that X_P^ℓ is a vertex cover of P. For every $i \in [4]$, all four edges incident to $a_{i,1}$ are covered and X_P^ℓ contains at least one vertex from the edge $\{c_{i,0}, c_{i,1}\}$. For every $i \in [4]$, all edges of the C_4 induced by $\{u_i, a_{i,3}, b_{i,0}, b_{i,1}\}$ are covered by X_P^ℓ, since X_P^ℓ picks one of the two antipodal pairs depending on $\mathtt{sol}(\mathbf{s}_i^\ell)$. The clique induced by $v_1, \ldots, v_6$ is fully covered by X_P^ℓ, since it picks five out of six vertices. The edges between v_ℓ and the indicator vertices are covered, because by construction of P, we have that $N(v_\ell) = (\{v_1, \ldots, v_6\} \setminus \{v_\ell\}) \cup \{b_{i,\mathtt{sol}(\mathbf{s}_i^\ell)} : i \in [4]\} \cup \{c_{i,\mathtt{conn}(\mathbf{s}_i^\ell)} : i \in [4]\} \subseteq X_P^\ell$. This shows that X_P^ℓ is a vertex cover of P. Very similar to the proof of Lemma 7.2.2, we see that $\mathbf{state}(X_P^\ell) = \mathbf{s}^\ell$.

It remains to show the property regarding connectivity. By assumption, only the join vertices $u_1, \ldots, u_4$ and clique vertices $v_1, \ldots, v_6$ can be adjacent to vertices outside of $P + \hat{r}$. However, as the clique vertices are neighbors of the root $\hat{r}$, their other connections to the outside of P cannot provide any new root-connectivity. If we have $\mathbf{state}_{X_P^\ell}(u_i) = \mathbf{1}_0$ for some $i \in [4]$, then u_i is root-connected in X via some path that leaves $P + \hat{r}$. Note that all vertices in $X_P^\ell \setminus \{u_i, a_{i,1} : i \in [4]\}$ are directly adjacent to the root $\hat{r}$, hence it just remains to handle the root-connectivity of $a_{i,1}$. If $\mathbf{state}_{X_P^\ell}(u_i) = \mathbf{0}$, then $a_{i,1}$ is root-connected via the path $a_{i,1}, b_{i,0}, \hat{r}$ in X. If $\mathbf{state}_{X_P^\ell}(u_i) = \mathbf{1}_0$, then we can extend the path that leaves P and connects u_i to $\hat{r}$ by $a_{i,1}$. Finally, if $\mathbf{state}_{X_P^\ell}(u_i) = \mathbf{1}_1$, then the path $a_{i,1}, c_{i,1}, \hat{r}$ exists in $X_P^\ell + \hat{r}$. This concludes the proof. □

State Transitions. In the lower bound construction, we will create long paths by repeatedly concatenating the path gadgets P. To study how the state can change between two consecutive path gadgets, suppose that we have two copies P^1 and P^2 of P such that the vertices u_3 and u_4 in P^1 are joined to the vertices u_1 and u_2 in P^2. We denote the vertices of P^1 with a superscript 1 and the vertices of P^2 with a superscript 2, e.g., u_3^1 refers to the vertex u_3 of P^1. Again, suppose that P^1 and P^2 are embedded as induced subgraphs in a larger graph G with a root vertex $\hat{r}$ and that only the vertices $u_1^1, u_2^1, u_3^2, u_4^2$ and the clique vertices v_ℓ^1, v_ℓ^2, $\ell \in [6]$, have neighbors outside of $P^1 + P^2 + \hat{r}$. Furthermore, X denotes a connected vertex cover with $\hat{r} \in X$.

Using the previous lemmas, we now show that states can only transition from one path gadget to the next according to the transition order and that it is also feasible for the state to remain stable.

Lemma 7.2.4. *Suppose that* $|X \cap P^1| \leq 21$ *and* $|X \cap P^2| \leq 21$*, then* $\mathbf{state}(X \cap P^1) = \mathbf{s}^{\ell_1}$ *and* $\mathbf{state}(X \cap P^2) = \mathbf{s}^{\ell_2}$ *with* $\ell_1 \leq \ell_2$.

Additionally, for each $\ell \in [6]$*, the set* $X^\ell = X^\ell_{P^1} \cup X^\ell_{P^2}$ *is a vertex cover of* $P^1 + P^2$ *with* $\mathbf{state}_{X^\ell}(\{u^1_3, u^1_4, u^2_1, u^2_2\}) \subseteq \{\mathbf{0}, \mathbf{1}_1\}$.

Proof. By Lemma 7.2.2, we have that $\mathbf{state}(X \cap P^1) = \mathbf{s}^{\ell_1}$ and $\mathbf{state}(X \cap P^2) = \mathbf{s}^{\ell_2}$ for some $\ell_1 \in [6]$ and $\ell_2 \in [6]$. It remains to show that $\ell_1 \leq \ell_2$.

Define $U^1 = \{u^1_3, u^1_4\}$, $U^2 = \{u^2_1, u^2_2\}$, and $U = U^1 \cup U^2$. By assumption only the clique vertices of P^1 and P^2 and the vertices $u^1_1, u^1_2, u^2_3, u^2_4$ are allowed to have neighbors outside of $P^1 + P^2 + \hat{r}$, hence $\{v^i_\ell : i \in [2], \ell \in [6]\} \subseteq N(\hat{r})$ separates the vertices in U from the root $\hat{r}$ in the whole graph G. Hence, we can see whether the vertices of $X \cap U$ are root-connected in X by just considering the graph $P^1 + P^2 + \hat{r}$.

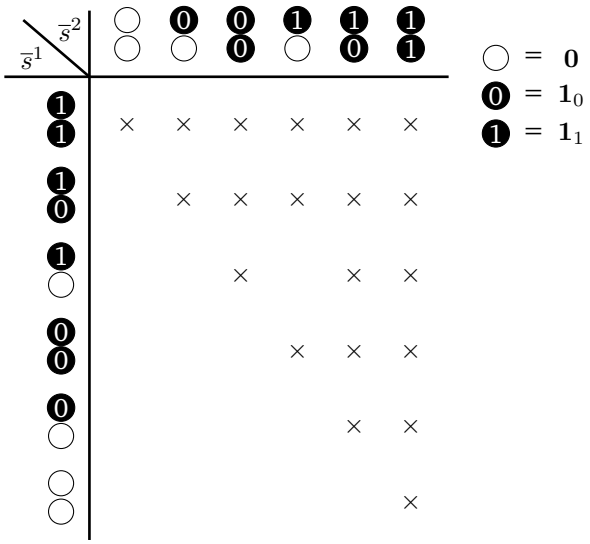


Fig. 7.3.: A matrix depicting the possible state transitions between two consecutive path gadgets. The rows are labeled with $\bar{\mathbf{s}}^1 = (\mathbf{s}^{\ell_1}_3, \mathbf{s}^{\ell_1}_4)$, $\ell_1 \in [6]$, and the columns are labeled with $\bar{\mathbf{s}}^2 = (\mathbf{s}^{\ell_2}_1, \mathbf{s}^{\ell_2}_2)$, $\ell_2 \in [6]$. An $\times$ marks possible state transitions in a connected vertex cover.

Define the atomic state pairs $\bar{\mathbf{s}}^1 = (\mathbf{s}^{\ell_1}_3, \mathbf{s}^{\ell_1}_4) = (\mathbf{state}_{X \cap P^1}(u^1_3), \mathbf{state}_{X \cap P^1}(u^1_4))$ and $\bar{\mathbf{s}}^2 = (\mathbf{s}^{\ell_2}_1, \mathbf{s}^{\ell_2}_2) = (\mathbf{state}_{X \cap P^2}(u^2_1), \mathbf{state}_{X \cap P^2}(u^2_2))$. We claim that X does not cover some edge in $G[U]$ or some vertex in $X \cap U$ is not root-connected in X whenever $\ell_1 > \ell_2$, see also Figure 7.3. Some edge in $G[U]$ is not covered by X if and only if both $\bar{\mathbf{s}}^1$ and $\bar{\mathbf{s}}^2$ contain a $\mathbf{0}$. Hence, we have that $(\ell_1, \ell_2) \notin \{(3,1), (3,2), (5,1), (5,2), (5,4), (6,1), (6,2), (6,4)\}$. Some vertex in $X \cap U$ is not root-connected in X if and only if both $\bar{\mathbf{s}}^1$ and $\bar{\mathbf{s}}^2$ contain no $\mathbf{1}_1$s at all or one consists of two $\mathbf{0}$s and the other one contains a $\mathbf{1}_0$. This additionally shows that $(\ell_1, \ell_2) \notin \{(2,1), (4,1), (4,2), (4,3), (5,3), (6,3), (6,5)\}$ and concludes the proof of the first part.

For the second part, notice that $\mathbf{state}(X^\ell_{P^1}) = \mathbf{state}(X^\ell_{P^2}) = \mathbf{s}^\ell$ by Lemma 7.2.3 and by the same approach as in the last paragraph, we see that for $\ell = \ell_1 = \ell_2$ all edges in $G[U]$ are

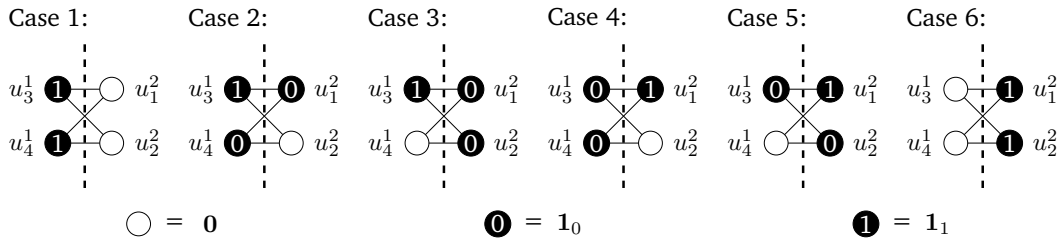


Fig. 7.4.: Case distinction to show $\mathbf{state}_{X^\ell}(\{u_3^1, u_4^1, u_1^2, u_2^2\}) \subseteq \{\mathbf{0}, \mathbf{1}_1\}$. The left side in each case depicts $\bar{\mathbf{s}}^1$ and the right side depicts $\bar{\mathbf{s}}^2$.

covered and all vertices in $X^\ell \cap U$ are root-connected in $X^\ell \cup \{\hat{r}\}$, see Figure 7.4 for the different cases. □

We say that a *cheat occurs* if $\ell_1 < \ell_2$. Creating arbitrarily long paths of the path gadgets P, Lemma 7.2.4 shows that at most $|\mathbf{States}| - 1 = 5 = \mathcal{O}(1)$ cheats occur on such a path which allows us to find a *cheat-free region* as outlined previously.

7.2.2 Complete Construction

Setup. Assume that Connected Vertex Cover can be solved in time $\mathcal{O}^*((6-\varepsilon)^{\text{lin-cw}(G)})$ for some $\varepsilon > 0$. Given a Satisfiability-instance σ with n variables and m clauses, we construct an equivalent Connected Vertex Cover instance with linear-clique-width approximately $n \log_6(2)$ so that the existence of such an algorithm for Connected Vertex Cover would imply that SETH is false.

We pick an integer β only depending on ε; the precise choice of β will be discussed at a later point. The variables of σ are partitioned into groups of size at most β, resulting in $t = \lceil n/\beta \rceil$ groups. Furthermore, we pick the smallest integer p that satisfies $6^p \geq 2^\beta$, i.e., $p = \lceil \log_6(2^\beta) \rceil$. We now begin with the construction of the Connected Vertex Cover instance $(G = G(\sigma, \beta), \bar{b})$.

We create the root vertex $\hat{r}$ and attach a leaf $\hat{r}'$ which forces $\hat{r}$ into any connected vertex cover. For every group $i \in [t]$, we create p long path-like gadgets $P^{i,j}$, $j \in [p]$, where each $P^{i,j}$ consists of $m(5tp+1)$ copies $P^{i,j,\ell}$, $\ell \in [m(5tp+1)]$, of the path gadget P and consecutive copies are connected by a join. More precisely, the vertices in some $P^{i,j,\ell}$ inherit their names from P and the superscript of $P^{i,j,\ell}$ and for every $i \in [t]$, $j \in [p]$, $\ell \in [m(5tp+1)-1]$, the vertices $\{u_3^{i,j,\ell}, u_4^{i,j,\ell}\}$ are joined to the vertices $\{u_1^{i,j,\ell+1}, u_2^{i,j,\ell+1}\}$. The ends of each path $P^{i,j}$, i.e. the vertices $u_1^{i,j,1}, u_2^{i,j,1}, u_3^{i,j,m(5tp+1)}, u_4^{i,j,m(5tp+1)}$, are made adjacent to the root $\hat{r}$.

For every group $i \in [t]$ and column $\ell \in [m(5tp+1)]$, we create a *decoding gadget* $D^{i,\ell}$ in the same style as Cygan et al. [49] for Connected Vertex Cover[pathwidth]. Every variable group i has at most 2^β possible truth assignments and by choice of p we have that $6^p \geq 2^\beta$, so there is an injective mapping $\kappa\colon \{0,1\}^\beta \to [6]^p$ which assigns to each truth assignment $\tau \in \{0,1\}^\beta$ a sequence $\kappa(\tau) \in [6]^p$. For each sequence $\mathbf{h} = (h_1, \ldots, h_p) \in [6]^p$, we create vertices $x_{\mathbf{h}}^{i,\ell}$, $\bar{x}_{\mathbf{h}}^{i,\ell}$, $y_{\mathbf{h}}^{i,\ell}$ and edges $\{x_{\mathbf{h}}^{i,\ell}, \bar{x}_{\mathbf{h}}^{i,\ell}\}$, $\{x_{\mathbf{h}}^{i,\ell}, y_{\mathbf{h}}^{i,\ell}\}$, $\{y_{\mathbf{h}}^{i,\ell}, \hat{r}\}$. We add the edge $\{x_{\mathbf{h}}^{i,\ell}, v_{h_j}^{i,j,\ell}\}$ for all $\mathbf{h} = (h_1, \ldots, h_p) \in [6]^p$ and $j \in [p]$. Finally, we create two adjacent vertices $z^{i,\ell}$ and $\bar{z}^{i,\ell}$ and edges $\{z^{i,\ell}, y_{\mathbf{h}}^{i,\ell}\}$ for all $\mathbf{h} \in [6]^p$. Each decoding gadget $D^{i,\ell}$ together

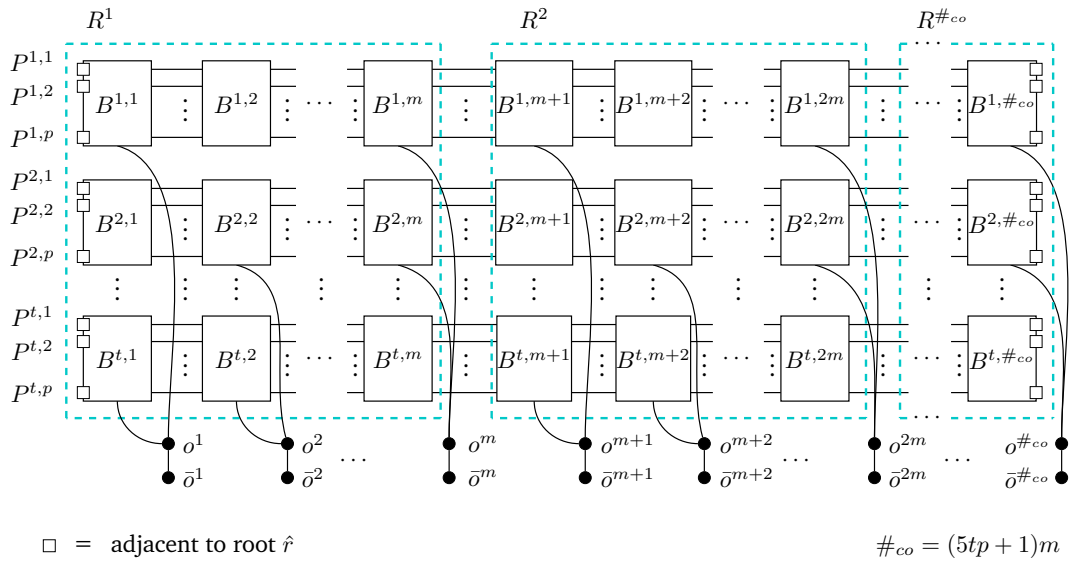


Fig. 7.5.: High-level construction for CONNECTED VERTEX COVER[linear-clique-width]. Every edge between two blocks $B^{i,\ell}$ and $B^{i,\ell+1}$ represents a join. The regions R^γ, highlighted by the dashed cyan rectangles, are important for the proof of Lemma 7.2.6.

with the adjacent path gadgets $P^{i,j,\ell}$, $j \in [p]$, forms the *block* $B^{i,\ell}$, see Figure 7.5 for a high-level depiction of the connections between different blocks.

Lastly, we construct the *clause gadgets*. We number the clauses of σ by $C_0, \ldots, C_{m-1}$. For every column $\ell \in [m(5tp+1)]$, we create an adjacent pair of vertices o^ℓ and $\bar{o}^\ell$. Let $\ell' \in [0, m-1]$ be the remainder of $(\ell - 1)$ modulo m; for all $i \in [t]$, $\mathbf{h} \in \kappa(\{0,1\}^\beta)$ such that $\kappa^{-1}(\mathbf{h})$ is a truth assignment for variable group i satisfying clause $C_{\ell'}$, we add the edge $\{o^\ell, y^{i,\ell}_{\mathbf{h}}\}$. See Figure 7.6 for a depiction of the decoding and clause gadgets and the connections to the path gadgets.

Lemma 7.2.5. *If σ is satisfiable, then there exists a connected vertex cover X of $G = G(\sigma, \beta)$ of size $|X| \leq (21tp + (6^p + 2)t + 1)m(5tp+1) + 1 = \overline{b}$.*

Proof. Let τ be a satisfying truth assignment of σ and let τ^i denote the restriction of τ to the i-th variable group for every $i \in [t]$ and let $\kappa(\tau^i) = \mathbf{h}^i = (h^i_1, \ldots, h^i_p) \in [6]^p$ be the corresponding sequence. The connected vertex cover is given by

$$X = \{\hat{r}\} \cup \bigcup_{\ell \in [m(5tp+1)]} \left(\{o^\ell\} \cup \bigcup_{i \in [t]} \left(\{y^{i,\ell}_{\mathbf{h}^i}, z^{i,\ell}\} \cup \bigcup_{\mathbf{h} \in [6]^p} \{x^{i,\ell}_{\mathbf{h}}\} \cup \bigcup_{j \in [p]} X^{h^i_j}_{P^{i,j,\ell}} \right) \right),$$

where $X^{h^i_j}_{P^{i,j,\ell}}$ refers to the sets given by Lemma 7.2.3.

Clearly, $|X| = \overline{b}$, so it remains to prove that X is a connected vertex cover. By Lemma 7.2.3 and the second part of Lemma 7.2.4 all edges induced by the path gadgets are covered by X and all vertices on the path gadgets that belong to X are root-connected, except for possibly the vertices at the ends, i.e. $\bigcup_{i \in [t]} \bigcup_{j \in [p]} \{u^{i,j,1}_1, u^{i,j,1}_2, u^{i,j,m(5tp+1)}_3, u^{i,j,m(5tp+1)}_4\}$, but these are contained in the neighborhood of $\hat{r}$ by construction of G.

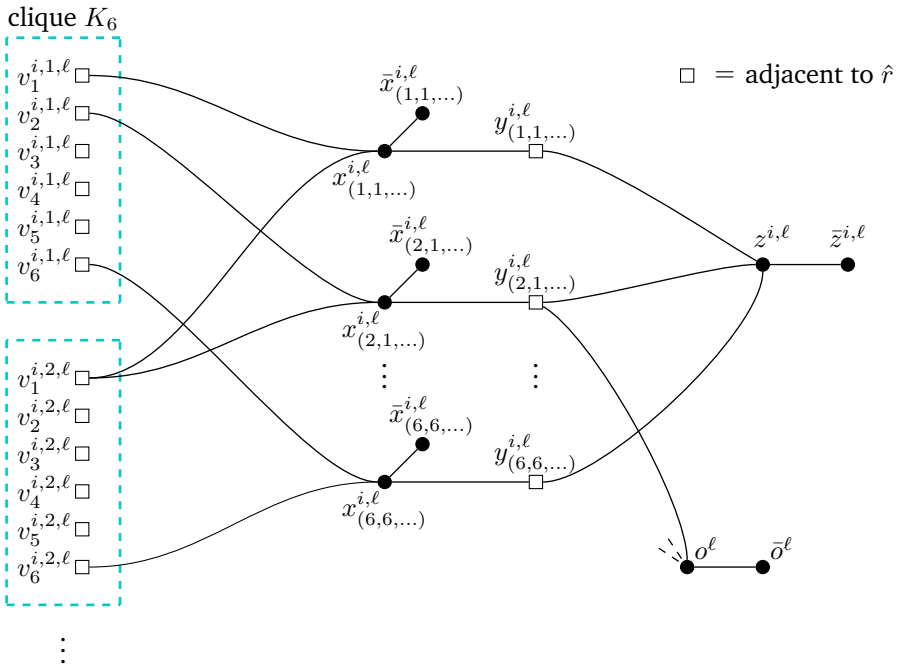


Fig. 7.6.: Decoding and clause gadget for Connected Vertex Cover[linear-clique-width].

Fix $i \in [t]$, $\ell \in [m(5tp+1)]$, and consider the corresponding decoding gadget. Since $z^{i,\ell} \in X$ and $x^{i,\ell}_{\mathbf{h}} \in X$ for all $\mathbf{h} \in [6]^p$, all edges induced by the decoding gadget and all edges between the decoding gadget and the path gadgets are covered by X. Furthermore, since $o^{\ell} \in X$, all edges inside the clause gadget and all edges between the clause gadget and the decoding gadgets are covered by X. Hence, X has to be a vertex cover of G.

It remains to prove that the vertices in the decoding and clause gadgets that belong to X are also root-connected. Again, fix $i \in [t]$, $\ell \in [m(5tp+1)]$, and $\mathbf{h} = (h_1, \ldots, h_p) \in [6]^p \setminus \{\mathbf{h}^i\}$. Since $\mathbf{h} \neq \mathbf{h}^i$, there is some $j \in [p]$ such that $h_j \neq h^i_j$ and hence $v^{i,j,\ell}_{h_j} \in X$ by Lemma 7.2.3 which connects $x^{i,\ell}_{\mathbf{h}}$ to the root $\hat{r}$. The vertices $x^{i,\ell}_{\mathbf{h}^i}$ and $z^{i,\ell}$ are root-connected via $y^{i,\ell}_{\mathbf{h}^i} \in X$.

We conclude by showing that o^{ℓ} is root-connected for all $\ell \in [m(5tp+1)]$. Since τ is a satisfying truth assignment of σ, there is some variable group $i \in [t]$ such that τ^i already satisfies clause $C_{\ell'}$, where ℓ' is the remainder of $(\ell - 1)$ modulo m. By construction of G and X, the vertex $y^{i,\ell}_{\mathbf{h}^i} \in X$ is adjacent to o^{ℓ}, since $\kappa(\tau^i) = \mathbf{h}^i$, and connects o^{ℓ} to the root $\hat{r}$. This shows that all vertices of X are root-connected, so $G[X]$ has to be connected. □

Lemma 7.2.6. *If there exists a connected vertex cover X of $G = G(\sigma, \beta)$ of size $|X| \leq (21tp + (6^p + 2)t + 1)m(5tp+1) + 1 = \bar{b}$, then σ is satisfiable.*

Proof. We begin by arguing that X has to satisfy $|X| = \bar{b}$. First, we must have that $\hat{r} \in X$, because $\hat{r}$ has a neighbor of degree 1. By Lemma 7.2.1, we have that $|X \cap P^{i,j,\ell}| \geq 21$ for all $i \in [t]$, $j \in [p]$, $\ell \in [m(5tp+1)]$. In every decoding gadget, i.e. one for every $i \in [t]$ and $\ell \in [m(5tp+1)]$, the set $\{z^{i,\ell}\} \cup \bigcup_{\mathbf{h} \in [6]^p} x^{i,\ell}_{\mathbf{h}}$ has to be contained in X, since every vertex in this set has a neighbor of degree 1. Furthermore, to connect $z^{i,\ell}$ to $\hat{r}$, at least one of the vertices $y^{i,\ell}_{\mathbf{h}}$, $\mathbf{h} \in [6]^p$, has to be contained in X. Hence, X must contain at least $6^p + 2$ vertices per decoding gadget. Lastly, $o^{\ell} \in X$ for all $\ell \in [m(5tp+1)]$, since o^{ℓ} has a neighbor

of degree 1. Since we have only considered disjoint vertex sets, this shows that $|X| = \bar{b}$ and all of the previous inequalities have to be tight.

By Lemma 7.2.2, we know that X assumes one of the six possible states on each $P^{i,j,\ell}$. Fix some $P^{i,j} = \bigcup_{\ell \in [m(5tp+1)]} P^{i,j,\ell}$ and note that due to Lemma 7.2.4 the state can change at most five times along $P^{i,j}$. Such a state change is called a *cheat*. Let $\gamma \in [0, 5tp]$ and define the γ-th *region* $R^\gamma = \bigcup_{i \in [t]} \bigcup_{j \in [p]} \bigcup_{\ell=\gamma m+1}^{(\gamma+1)m} P^{i,j,\ell}$. Since there are $5tp+1$ regions, there is at least one region R^γ such that no cheat occurs in R^γ. We will consider this region for the remainder of the proof and read off a satisfying truth assignment from this region.

For $i \in [t]$, define $\mathbf{h}^i = (h^i_1, \ldots, h^i_p) \in [6]^p$ such that $v^{i,j,\gamma m+1}_{h^i_j} \notin X$ for all $j \in [p]$; this is well-defined by Lemma 7.2.2. Since R^γ does not contain any cheats, the definition of $\mathbf{h}^i$ is independent of which column $\ell \in [\gamma m + 1, (\gamma+1)m]$ we consider. For every $i \in [t]$ and $\ell \in [\gamma m + 1, (\gamma+1)m]$, we claim that $y^{i,\ell}_{\mathbf{h}} \in X$ if and only if $\mathbf{h} = \mathbf{h}^i$. We have already established that for every i and ℓ, there is exactly one $\mathbf{h}$ such that $y^{i,\ell}_{\mathbf{h}} \in X$. Consider the vertex $x^{i,\ell}_{\mathbf{h}^i} \in X$, its neighbors in G are $v^{i,1,\ell}_{h^i_1}, v^{i,2,\ell}_{h^i_2}, \ldots, v^{i,p,\ell}_{h^i_p}$, $\bar{x}^{i,\ell}_{\mathbf{h}^i}$, and $y^{i,\ell}_{\mathbf{h}^i}$. By construction of $\mathbf{h}^i$ and the tight allocation of the budget, we have $(N_G(x^{i,\ell}_{\mathbf{h}^i}) \setminus \{y^{i,\ell}_{\mathbf{h}^i}\}) \cap X = \emptyset$. Therefore, X has to include $y^{i,\ell}_{\mathbf{h}^i}$ to connect $x^{i,\ell}_{\mathbf{h}^i}$ to the root $\hat{r}$. This shows the claim.

For $i \in [t]$, we define the truth assignment τ^i for variable group i by taking an arbitrary truth assignment if $\mathbf{h}^i \notin \kappa(\{0,1\}^\beta)$ and setting $\tau^i = \kappa^{-1}(\mathbf{h}^i)$ otherwise. By setting $\tau = \bigcup_{i \in [t]} \tau^i$ we obtain a truth assignment for all variables and we claim that τ satisfies σ. Consider some clause $C_{\ell'}$, $\ell' \in [0, m-1]$, and let $\ell = \gamma m + \ell' + 1$. We have already argued that $o^\ell \in X$ and to connect o^ℓ to the root $\hat{r}$, there has to be some $y^{i,\ell}_{\mathbf{h}} \in N_G(o^\ell) \cap X$. By the previous claim, $\mathbf{h} = \mathbf{h}^i$ and therefore τ^i, and also τ, satisfy clause $C_{\ell'}$ due to the construction of G. Because the choice of $C_{\ell'}$ was arbitrary, τ has to be a satisfying assignment of σ. □

Lemma 7.2.7. *The constructed graph $G = G(\sigma, \beta)$ has* $\text{lin-cw}(G) \leq tp + 3 \cdot 6^p + \mathcal{O}(1)$ *and a linear clique-expression of this width can be constructed in polynomial time.*

Proof. We will describe how to construct a linear clique-expression for G of width $tp + 3 \cdot 6^p + \mathcal{O}(1)$. The clique-expression will use one path label (i, j) for every long path $P^{i,j}$, $i \in [t]$, $j \in [p]$, temporary decoding labels for every vertex of a decoding gadget, i.e. $3 \cdot 6^p + 2$ many, temporary gadget labels for every vertex of a path gadget, i.e. 38 many, two temporary clause labels for the clause gadget, one label for the root vertex and a trash label.

The construction of the clique-expression is described in Algorithm 3 and the central idea is to proceed column by column and group by group in each column. By reusing the temporary labels, we keep the total number of labels small. The maximum number of labels used simultaneously occurs in line 7 and is $tp + (3 \cdot 6^p + 2) + 38 + 2 + 1 + 1$. This concludes the proof. □

Theorem 7.2.8. *There is no algorithm that solves* Connected Vertex Cover*, given a linear k-expression, in time $\mathcal{O}^*((6-\varepsilon)^k)$ for some $\varepsilon > 0$, unless CNF-SETH fails.*

Proof. Assume that there exists an algorithm $\mathbb{A}$ that solves Connected Vertex Cover in time $\mathcal{O}^*((6-\varepsilon)^k)$ for some $\varepsilon > 0$ given a linear k-expression. Given β, we define $\delta_1 < 1$ such that $(6-\varepsilon)^{\log_6(2)} = 2^{\delta_1}$ and δ_2 such that $(6-\varepsilon)^{1/\beta} = 2^{\delta_2}$. By picking β large enough, we can ensure that $\delta = \delta_1 + \delta_2 < 1$. We will show how to solve Satisfiability using $\mathbb{A}$ in

Algorithm 3: Constructing a linear clique-expression for G.

Introduce $\hat{r}$ with root label and $\hat{r}'$ with trash label and add the edge $\{\hat{r}, \hat{r}'\}$;
for $\ell \in [m(5tp+1)]$ **do**
  Introduce o^ℓ and $\bar{o}^\ell$ and the edge $\{o^\ell, \bar{o}^\ell\}$ with the clause labels;
  for $i \in [t]$ **do**
    Build $D^{i,\ell}$ using decoding labels and add edges to o^ℓ;
    for $j \in [p]$ **do**
      Build $P^{i,j,\ell}$ using gadget labels and add edges to $D^{i,j}$;
      Join $u_1^{i,j,\ell}$ and $u_2^{i,j,\ell}$ to path label (i,j) if $\ell > 1$ and to root label otherwise;
      Relabel path label (i,j) to trash label;
      Relabel $u_3^{i,j,\ell}$ and $u_4^{i,j,\ell}$ to path label (i,j);
      Relabel all other gadget labels to the trash label;
    Relabel all decoding labels to the trash label;
  Relabel all clause labels to the trash label;

time $\mathcal{O}^*(2^{\delta n}\operatorname{poly}(m)) = \mathcal{O}^*(2^{\delta n})$, where n is the number of variables, thus contradicting CNF-SETH, Conjecture 2.1.1.

Given a SATISFIABILITY instance σ, we construct the graph $G = G(\sigma, \beta)$ and the linear clique-expression from Lemma 7.2.7 in polynomial time, note that we have $\beta = \mathcal{O}(1)$ and hence $p = \mathcal{O}(1)$; recall $p = \lceil \log_6(2^\beta) \rceil$. We then run the assumed algorithm $\mathbb{A}$ on G and return its answer. This is correct by Lemma 7.2.5 and Lemma 7.2.6. Due to Lemma 7.2.7, the running time is

$$\begin{aligned}
& \mathcal{O}^*\left((6-\varepsilon)^{tp+3\cdot 6^p+\mathcal{O}(1)}\right) \leq \mathcal{O}^*\left((6-\varepsilon)^{tp}\right) && \leq \mathcal{O}^*\left((6-\varepsilon)^{\lceil \frac{n}{\beta} \rceil p}\right) \\
\leq\ & \mathcal{O}^*\left((6-\varepsilon)^{\frac{n}{\beta} p}\right) && \leq \mathcal{O}^*\left((6-\varepsilon)^{\frac{n}{\beta}\lceil \log_6(2^\beta)\rceil}\right) \leq \mathcal{O}^*\left((6-\varepsilon)^{\frac{n}{\beta}\log_6(2^\beta)}(6-\varepsilon)^{\frac{n}{\beta}}\right) \\
\leq\ & \mathcal{O}^*\left(2^{\delta_1 \beta \frac{n}{\beta}} 2^{\delta_2 n}\right) && \leq \mathcal{O}^*\left(2^{(\delta_1+\delta_2)n}\right) && \leq \mathcal{O}^*\left(2^{\delta n}\right),
\end{aligned}$$

hence completing the proof. □

7.3 Connected Dominating Set Lower Bound

This subsection is devoted to proving that CONNECTED DOMINATING SET (with unit costs) cannot be solved in time $\mathcal{O}^*((5-\varepsilon)^{\text{lin-cw}(G)})$ for some $\varepsilon > 0$ unless the CNF-SETH, Conjecture 2.1.1, fails. We briefly outline the intuition behind the design of the path gadget, which largely follows the same approach as for CONNECTED VERTEX COVER. Afterwards, we present the construction of the path gadget and analyze it, then we move on to the complete construction and correctness proofs. The decoding gadgets are again directly adapted from the lower bound for CONNECTED VERTEX COVER[pathwidth] given by Cygan et al. [49].

Root. We create a distinguished vertex $\hat{r}$ called the *root* and by attaching a vertex of degree 1 to $\hat{r}$ we ensure that every connected dominating set has to contain $\hat{r}$.

	$\{\mathbf{0}_1, \mathbf{0}_0\}$	$\{\mathbf{0}_1\}$	$\{\mathbf{1}_0, \mathbf{0}_0\}$	$\{\mathbf{1}_1, \mathbf{0}_0\}$	$\{\mathbf{1}_1\}$
$\{\mathbf{1}_1\}$	1	1	1	1	1
$\{\mathbf{0}_1\}$	0	1	0	0	1
$\{\mathbf{1}_1, \mathbf{0}_0\}$	0	0	1	1	1
$\{\mathbf{1}_0, \mathbf{0}_0\}$	0	0	0	1	1
$\{\mathbf{0}_1, \mathbf{0}_0\}$	0	0	0	0	1

Tab. 7.3.: A largest triangular submatrix, after reordering rows and columns, of the compatibility matrix for CONNECTED DOMINATING SET.

7.3.1 Path Gadget

To rule out the running time $\mathcal{O}^*((5-\varepsilon)^{\text{lin-cw}(G)})$ for any $\varepsilon > 0$, we have to build a path gadget that admits 5 distinct states and narrows down to a single label, so that each row of the construction contributes one unit of linear-clique-width. We begin by analyzing the possible behaviors of a partial solution on a label.

First, we consider the possible states of a single vertex v with respect to a partial solution X. Compared to CONNECTED VERTEX COVER, there is one more state, as the state $\mathbf{0}$ splits into the states $\mathbf{0}_1$ and $\mathbf{0}_0$, which denote that $v \notin X$ and whether v is dominated by X or not, and we keep the states $\mathbf{1}_1$ and $\mathbf{1}_0$. Hence, the state of a label can be represented by a subset of $\{\mathbf{0}_0, \mathbf{0}_1, \mathbf{1}_0, \mathbf{1}_1\}$.

Similar to before, we study the compatibility of these label states across a join. The result is a matrix of size 15×15, as we can exclude the empty subset. However, many states lead to the same compatibility pattern, e.g. for any subset $\emptyset \neq \mathbf{S} \subseteq \{\mathbf{0}_0, \mathbf{0}_1, \mathbf{1}_0, \mathbf{1}_1\}$ the states $\mathbf{S}$ and $\mathbf{S} \cup \{\mathbf{0}_1\}$ yield the same compatibility pattern, since the vertex state $\mathbf{0}_1$ does not add any additional constraint. It turns out that the compatibility matrix contains only five distinct rows, which give rise to the triangular submatrix Table 7.3 of size 5×5 after reordering. Surprisingly, the number of redundancies is so large that, although CONNECTED DOMINATING SET has four vertex states compared to only three for CONNECTED VERTEX COVER, we end up with fewer label states than for CONNECTED VERTEX COVER.

Again, it is sufficient to take independent sets of size two as labels incident to the join. The relevant label states will be represented by the following ordered pairs of vertex states: $(\mathbf{0}_1, \mathbf{0}_0)$, $(\mathbf{0}_1, \mathbf{0}_1)$, $(\mathbf{1}_0, \mathbf{0}_0)$, $(\mathbf{1}_1, \mathbf{0}_0)$, $(\mathbf{1}_1, \mathbf{1}_1)$. By pairing these states along the diagonal of the triangular submatrix, we then obtain the desired states for the path gadget in the transition order.

Formal Definition of States. To capture the vertex states, we define the four atomic states $\mathbf{Atoms} = \{\mathbf{0}_0, \mathbf{0}_1, \mathbf{1}_0, \mathbf{1}_1\}$ and the predicates $\texttt{sol}, \texttt{conn}, \texttt{dom}\colon \mathbf{Atoms} \to \{0,1\}$ by $\texttt{sol}(\mathbf{a}) = [\mathbf{a} \in \{\mathbf{1}_0, \mathbf{1}_1\}]$, $\texttt{conn}(\mathbf{a}) = [\mathbf{a} = \mathbf{1}_1]$, and $\texttt{dom}(\mathbf{a}) = [\mathbf{a} = \mathbf{0}_1]$. The atoms $\mathbf{0}_1$ and $\mathbf{0}_0$ mean that a vertex is not inside the partial solution and the subscript denotes whether the vertex is dominated by the partial solution or not; $\mathbf{1}_1$ and $\mathbf{1}_0$ indicate that a vertex is inside

the partial solution and the subscript indicates whether it is root-connected or not. Building on these atomic states, we define five states consisting of four atomic states each:

$$\begin{aligned}\mathbf{s}^1 &= (\mathbf{0}_1, \mathbf{0}_0, \mathbf{1}_1, \mathbf{0}_1)\\ \mathbf{s}^2 &= (\mathbf{0}_1, \mathbf{0}_1, \mathbf{0}_1, \mathbf{0}_1)\\ \mathbf{s}^3 &= (\mathbf{1}_0, \mathbf{0}_0, \mathbf{1}_1, \mathbf{0}_0)\\ \mathbf{s}^4 &= (\mathbf{1}_1, \mathbf{0}_0, \mathbf{1}_0, \mathbf{0}_0)\\ \mathbf{s}^5 &= (\mathbf{1}_1, \mathbf{0}_1, \mathbf{0}_1, \mathbf{0}_0)\end{aligned}$$

We collect these states into the set $\mathbf{States} = \{\mathbf{s}^1, \ldots, \mathbf{s}^5\}$ and use the notation $\mathbf{s}^\ell_i \in \mathbf{Atoms}$, $\ell \in [5]$, $i \in [4]$, to refer to the i-th atomic component of state $\mathbf{s}^\ell$. Note that $\mathbf{s}^5$ can be obtained from $\mathbf{s}^1$ by swapping the first two components with the last two components; in the same way, $\mathbf{s}^4$ can be obtained from $\mathbf{s}^3$.

Given a partial solution $Y \subseteq V(G)$, we associate to each vertex its state in Y with the map $\mathbf{state}_Y \colon V(G) \setminus \{\hat{r}\} \to \mathbf{Atoms}$, which is defined by

$$\mathbf{state}_Y(v) = \begin{cases} \mathbf{0}_0 & \text{if } v \notin N[Y \cup \{\hat{r}\}],\\ \mathbf{0}_1 & \text{if } v \in N[Y \cup \{\hat{r}\}] \setminus Y,\\ \mathbf{1}_0 & \text{if } v \in Y \text{ and } v \text{ is not root-connected in } Y \cup \{\hat{r}\},\\ \mathbf{1}_1 & \text{if } v \in Y \text{ and } v \text{ is root-connected in } Y \cup \{\hat{r}\}.\end{cases}$$

Subdivided Edges. In the graph construction, we frequently need *subdivided edges*. Given two vertices u and v, adding a *subdivided edge* between u and v means adding a new vertex $w_{\{u,v\}}$ and the edges $\{u, w_{\{u,v\}}\}$ and $\{w_{\{u,v\}}, v\}$. The crucial property of a subdivided edge between u and v is that any connected dominating set X has to contain at least one of u and v, since u and v remain the only neighbors of $w_{\{u,v\}}$ throughout the entire construction. In this sense, connected dominating sets behave in regards to subdivided edges as vertex covers do to normal edges, hence allowing us to adapt a substantial part of the construction from Connected Vertex Cover to Connected Dominating Set.

Formal Construction. We proceed by describing how to construct the path gadget P. We create the following sets of vertices:

- 4 *join* vertices $u_{1,1}, u_{1,2}, u_{2,1}, u_{2,2}$,
- 6 *auxiliary* vertices $a_{1,1}, a_{1,2}, a_{1,3}, a_{2,1}, a_{2,2}, a_{2,3}$,
- 4 *solution indicator* vertices $b_{1,0}, b_{1,1}, b_{2,0}, b_{2,1}$,
- 4 *connectivity indicator* vertices $c_{1,0}, c_{1,1}, c_{2,0}, c_{2,1}$,
- 4 *domination indicator* vertices $d_{1,0}, d_{1,1}, d_{2,0}, d_{2,1}$, and
- 5 *clique* vertices $v_1, \ldots, v_5$.

For every $i \in [2]$, we add the edges $\{u_{i,1}, a_{i,1}\}$, $\{a_{i,1}, a_{i,2}\}$, $\{a_{i,1}, b_{i,0}\}$, $\{a_{i,1}, c_{i,1}\}$, and $\{u_{i,2}, d_{i,1}\}$. Moreover, for every $i \in [2]$, we add subdivided edges from $u_{i,1}$ to $a_{i,3}$ and $b_{i,0}$; from $a_{i,3}$ to $b_{i,1}$; from $b_{i,0}$ to $b_{i,1}$; from $c_{i,0}$ to $c_{i,1}$; and from $d_{i,0}$ to $d_{i,1}$.

We make the following vertices adjacent to the root vertex $\hat{r}$: the auxiliary vertices $a_{i,3}$ for all $i \in [2]$, all solution indicator vertices, all connectivity indicator vertices, all domination

indicator vertices, and all clique vertices. We add all possible subdivided edges between the clique vertices v_ℓ, $\ell \in [5]$, so that they induce a clique of size 5 where every edge is subdivided.

Finally, we explain how to connect the indicator vertices to the clique vertices. The clique vertex v_ℓ corresponds to choosing state $\mathbf{s}^\ell$ on the join vertices $(u_{1,1}, u_{1,2}, u_{2,1}, u_{2,2})$. The indicator vertices are supposed to describe the behavior of a partial solution X of $P + \hat{r}$ as follows:

- $b_{i,1} \in X \iff u_{i,1} \in X$,
- $c_{i,1} \in X \iff u_{i,1} \in X$ and $u_{i,1}$ is root-connected in $X \cup \{\hat{r}\}$,
- $d_{i,1} \in X \iff u_{i,2} \in N[X] \setminus X$, i.e., $u_{1,2}$ is dominated by X.

Accordingly, for all $i \in [2]$ and $\ell \in [5]$, we add subdivided edges from v_ℓ to $b_{i,\mathtt{sol}(\mathbf{s}^\ell_{2i-1})}$ and $c_{i,\mathtt{conn}(\mathbf{s}^\ell_{2i-1})}$ and $d_{i,\mathtt{dom}(\mathbf{s}^\ell_{2i})}$. This concludes the construction of P, see Figure 7.7 for a depiction of the construction.

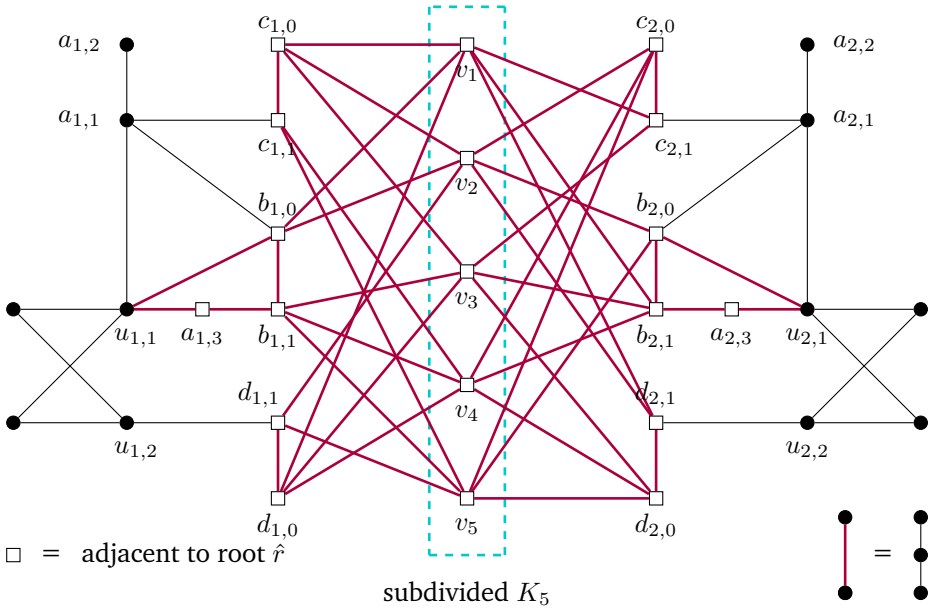


Fig. 7.7.: The path gadget for CONNECTED DOMINATING SET[linear-clique-width]. Vertices denoted by squares are adjacent to the root vertex $\hat{r}$ and bold red edges represent subdivided edges. The edges of the subdivided 5-clique formed by the clique vertices v_ℓ, $\ell \in [5]$, are not depicted. Besides the connections to $\hat{r}$, only the join vertices $u_{i,1}, u_{i,2}$, $i \in [2]$, and clique vertices v_ℓ, $\ell \in [5]$, have connections to outside this path gadget.

Behavior of a Single Path Gadget. For the upcoming lemmas, we assume that G is a graph that contains $P + \hat{r}$ as an induced subgraph and that only the join vertices $u_{i,1}, u_{i,2}, i \in [2]$, and clique vertices $v_\ell, \ell \in [5]$, have neighbors outside this copy of $P + \hat{r}$. Furthermore, let X be a connected dominating set of G with $\hat{r} \in X$. We study the behavior of such connected dominating sets on P; we will abuse notation and write $X \cap P$ instead of $X \cap V(P)$. The assumption on how P connects to the remaining graph implies that any vertex $v \in V(P)$ with $\mathbf{state}_{X \cap P}(v) = \mathbf{1}_0$ has to be root-connected in X through some path that leaves $P + \hat{r}$

via one of the join vertices $u_{i,j}$, $i,j \in [2]$. Note that any path leaving $P + \hat{r}$ through some clique vertex v_ℓ, $\ell \in [5]$, immediately yields a path to $\hat{r}$ in $P + \hat{r}$ as $\{v_\ell : \ell \in [5]\} \subseteq N(\hat{r})$.

We obtain a lower bound for $|X \cap P|$ via a vertex-disjoint packing of subgraphs.

Lemma 7.3.1. *Any connected dominating set X with $\hat{r} \in X$ satisfies $|X \cap P| \geq 14 = 2 \cdot 5 + 4$ and more specifically, $a_{i,1} \in X$, $|X \cap \{b_{i,0}, b_{i,1}, a_{i,3}, u_{i,1}\}| \geq 2$, $X \cap \{c_{i,0}, c_{i,1}\} \neq \emptyset$, $X \cap \{d_{i,0}, d_{i,1}\} \neq \emptyset$ for all $i \in [2]$ and $|X \cap \{v_1, \ldots, v_5\}| \geq 4$.*

Proof. First, observe that the closed neighborhoods of each of the following vertices are disjoint: $a_{i,2}$, $w_{\{b_{i,0},b_{i,1}\}}$, $w_{\{a_{i,3},u_{i,1}\}}$, $w_{\{c_{i,0},c_{i,1}\}}$, $w_{\{d_{i,0},d_{i,1}\}}$ for all $i \in [2]$. Since X is a dominating set, X contains at least one vertex of each of these neighborhoods. Since X is connected, it follows that $a_{i,1} \in X$, and for each of the subdivided edges one of the endpoints must be contained in X.

Next, we turn to the subdivided 5-clique. Again, X has to contain at least one endpoint of each subdivided edge present in this clique. If there were two clique vertices $v_i, v_j \notin X$, $i \neq j$, then the subdivided edge between them is not resolved by X. Therefore, X can avoid taking at most one of the clique vertices. □

For any partial solution $Y \subseteq V(P)$, we formalize the states at the boundary of P with the 4-tuple

$$\mathbf{state}(Y) = (\mathbf{state}_Y(u_{1,1}), \mathbf{state}_Y(u_{1,2}), \mathbf{state}_Y(u_{2,1}), \mathbf{state}_Y(u_{2,2})).$$

The following lemma shows that the states communicated to the boundary depend on the state of the central clique in the desired way.

Lemma 7.3.2. *Any connected dominating set X with $\hat{r} \in X$ and $|X \cap P| \leq 14$ satisfies $a_{i,1} \in X$, $X \cap \{b_{i,0}, b_{i,1}, a_{i,3}, u_{i,1}\} \in \{\{b_{i,0}, a_{i,3}\}, \{b_{i,1}, u_{i,1}\}\}$, $|X \cap \{c_{i,0}, c_{i,1}\}| = 1$, $|X \cap \{d_{i,0}, d_{i,1}\}| = 1$ for all $i \in [2]$ and there exists a unique $\ell \in [5]$ such that $v_\ell \notin X$. Furthermore, we have that $\mathbf{state}(X \cap P) = \mathbf{s}^\ell$.*

Proof. All the inequalities of Lemma 7.3.1 have to be tight in this case which proves everything of the first part except the part regarding $\{b_{i,0}, b_{i,1}, a_{i,3}, u_{i,1}\}$. We know that X contains exactly two of these vertices, but if X contains two that are connected by a subdivided edge, then the subdivided edge between the other two vertices is not resolved. Therefore, $X \cap \{b_{i,0}, b_{i,1}, a_{i,3}, u_{i,1}\} \in \{\{b_{i,0}, a_{i,3}\}, \{b_{i,1}, u_{i,1}\}\}$.

It remains to prove the property regarding the states of the join vertices. Since $v_\ell \notin X$, the other endpoints of the incident subdivided edges have to be contained in X. By construction, these are the vertices $b_{i,\mathtt{sol}(\mathbf{s}^\ell_{2i-1})}$, $c_{i,\mathtt{conn}(\mathbf{s}^\ell_{2i-1})}$, and $d_{i,\mathtt{dom}(\mathbf{s}^\ell_{2i})}$ for $i \in [2]$. Fix $i \in [2]$, and note that by the budget allocation inside P, it follows that $u_{i,2} \notin X$. Therefore, $u_{i,2}$ can only be dominated by $d_{i,1}$ inside P and $\mathbf{state}_{X \cap P}(u_{i,2}) = \mathbf{0}_1 \iff d_{i,1} \in X \iff \mathtt{dom}(\mathbf{s}^\ell_{2i}) = 1 \iff \mathbf{s}^\ell_{2i} = \mathbf{0}_1$. We have that $u_{i,1} \in X \iff b_{i,1} \in X \iff \mathtt{sol}(\mathbf{s}^\ell_{2i-1}) = 1 \iff \mathbf{s}^\ell_{2i-1} \in \{\mathbf{1}_0, \mathbf{1}_1\}$ and $u_{i,1}$ is always dominated by $a_{i,1}$, hence $\mathbf{state}_{X \cap P}(u_{i,1}) \neq \mathbf{0}_0$ in all cases. If $u_{i,1} \in X$, then its only neighbor inside $X \cap P$ is $a_{i,1}$ which brings root-connectivity to $u_{i,1}$ if and only if $c_{i,1} \in X$, therefore $\mathbf{state}_{X \cap P}(u_{i,1}) = \mathbf{1}_1 \iff c_{i,1} \in X \iff \mathtt{conn}(\mathbf{s}^\ell_{2i-1}) = 1 \iff \mathbf{s}^\ell_{2i-1} = \mathbf{1}_1$. This concludes the proof. □

Next, we establish that for any $\mathbf{s}^\ell \in \mathbf{States}$, a partial solution attaining $\mathbf{s}^\ell$ actually exists and that these partial solutions are root-connected and dominate everything inside P, if this holds at the join vertices $u_{i,j}$, $i,j \in [2]$.

Lemma 7.3.3. *For every $\ell \in [5]$, there exists a set X_P^ℓ of P such that $|X_P^\ell| = 14$, $X_P^\ell \cap \{v_1,\ldots,v_5\} = \{v_1,\ldots,v_5\} \setminus \{v_\ell\}$, $\mathbf{state}(X_P^\ell) = \mathbf{s}^\ell$, and*

$$\mathbf{state}_{X_P^\ell}(V(P) \setminus \{a_{1,1}, a_{2,1}, u_{1,1}, u_{1,2}, u_{2,1}, u_{2,2}\}) \subseteq \{\mathbf{0}_1, \mathbf{1}_1\}.$$

If X is a vertex subset of G with $\hat{r} \in X$ and $X \cap P = X_P^\ell$ and $\mathbf{state}_X(\{u_{i,1}, u_{1,2}\}) \subseteq \{\mathbf{0}_1, \mathbf{1}_1\}$ for every $i \in [2]$, then $\mathbf{state}_X(V(P)) \subseteq \{\mathbf{0}_1, \mathbf{1}_1\}$.

Proof. We define

$$\begin{aligned} X_P^\ell = {} & (\{v_1,\ldots,v_5\} \setminus \{v_\ell\}) \cup \{a_{i,1} : i \in [2]\} \\ & \cup \{u_{i,1} : i \in [2] \text{ and } \mathtt{sol}(\mathbf{s}_{2i-1}^\ell) = 1\} \cup \{a_{i,3} : i \in [2] \text{ and } \mathtt{sol}(\mathbf{s}_{2i-1}^\ell) = 0\} \\ & \cup \{b_{i,\mathtt{sol}(\mathbf{s}_{2i-1}^\ell)} : i \in [2]\} \cup \{c_{i,\mathtt{conn}(\mathbf{s}_{2i-1}^\ell)} : i \in [2]\} \cup \{d_{i,\mathtt{dom}(\mathbf{s}_{2i}^\ell)} : i \in [2]\} \end{aligned}$$

and claim that X_P^ℓ has the desired properties. We clearly have that $|X_P^\ell| = 14$ and $v_\ell \notin X_P^\ell$. We proceed by showing that every vertex of P with the 6 exceptions $a_{1,1}, a_{2,1}, u_{1,1}, u_{1,2}, u_{2,1}, u_{2,2}$ is dominated by X_P^ℓ or root-connected in $X_P^\ell + \hat{r}$.

First, observe that for any vertex $v \in N(\hat{r}) \cap V(P)$, we have that $\mathbf{state}_{X_P^\ell}(v) \in \{\mathbf{0}_1, \mathbf{1}_1\}$. Since $a_{i,1} \in X_P^\ell$ and $a_{i,2} \notin X_P^\ell$, we have that $\mathbf{state}_{X_P^\ell}(a_{i,2}) = \mathbf{0}_1$ for every $i \in [2]$. It remains to handle the subdividing vertices $w_{\{x,y\}}$. Since X_P^ℓ contains one vertex from every pair of indicator vertices, every subdividing edge between such a pair is resolved. By construction X_P^ℓ either contains the pair $\{b_{i,0}, a_{i,3}\}$ or the pair $\{b_{i,1}, u_{i,1}\}$ for every $i \in [2]$ and in either case the subdivided edges incident to $a_{i,3}$ and $u_{i,1}$ are resolved. The subdivided edges between the clique vertices are resolved because X_P^ℓ contains 4 out of 5 clique vertices. Finally, the subdivided edges between v_ℓ and the indicator vertices are resolved, since by construction of P, the subdivided edges incident to v_ℓ lead to $b_{i,\mathtt{sol}(\mathbf{s}_{2i-1}^\ell)}$, $c_{i,\mathtt{conn}(\mathbf{s}_{2i-1}^\ell)}$, and $d_{i,\mathtt{dom}(\mathbf{s}_{2i}^\ell)}$ for all $i \in [2]$ which are precisely the indicator vertices contained in X_P^ℓ.

By similar arguments as in the proof of Lemma 7.3.2, we obtain $\mathbf{state}(X_P^\ell) = \mathbf{s}^\ell$.

We proceed with the second part. By the first part, it remains to handle the vertices $a_{1,1}, a_{2,1}, u_{1,1}, u_{1,2}, u_{2,1}, u_{2,2}$. By assumption, the join-vertices that are not contained in X are dominated and those that are contained in X are root-connected. By definition of X_P^ℓ, we see that $a_{1,1}, a_{2,1} \in X$, hence it remains to establish the root-connectivity of $a_{1,1}$ and $a_{2,1}$.

We show that $a_{i,1}$ is root-connected by considering the different cases for $\mathbf{state}_{X_P^\ell}(u_{i,1})$. Note that $\mathbf{state}_{X_P^\ell}(u_{i,1}) \neq \mathbf{0}_0$, since $a_{i,1} \in X_P^\ell$. If $\mathbf{state}_{X_P^\ell}(u_{i,1}) = \mathbf{0}_1$, then $a_{i,1}$ is root-connected via the path $a_{i,1}, b_{i,0}, \hat{r}$ in X. If $\mathbf{state}_{X_P^\ell}(u_{i,1}) = \mathbf{1}_0$, then $u_{i,1}$ is root-connected in X via some path that leaves $P + \hat{r}$ which we can extend to $a_{i,1}$. Finally, if $\mathbf{state}_{X_P^\ell}(u_{i,1}) = \mathbf{1}_1$, then the path $a_{i,1}, c_{i,1}, \hat{r}$ exists in $G[X]$. □

State Transitions. In the lower bound construction, we again create long paths by repeatedly concatenating the path gadgets P. To study how the state can change between two consecutive path gadgets, suppose that we have two copies P^1 and P^2 of P such that the

vertices $u_{2,1}$ and $u_{2,2}$ in P^1 are joined to the vertices $u_{1,1}$ and $u_{1,2}$ in P^2. We denote the vertices of P^1 with a superscript 1 and the vertices of P^2 with a superscript 2, e.g., $u^1_{2,1}$ refers to the vertex $u_{2,1}$ of P^1. Again, suppose that P^1 and P^2 are embedded as induced subgraphs in a larger graph G with a root vertex $\hat{r}$ and that only the vertices $u^1_{1,1}, u^1_{1,2}, u^2_{2,1}, u^2_{2,2}$ and the clique vertices v^1_ℓ, v^2_ℓ, $\ell \in [5]$, have neighbors outside of $P^1 + P^2 + \hat{r}$. Furthermore, X denotes a connected dominating set of G with $\hat{r} \in X$.

The previous lemmas allow us to conclude that the gadget state can indeed only transition according to the transition order and that the state can remain stable.

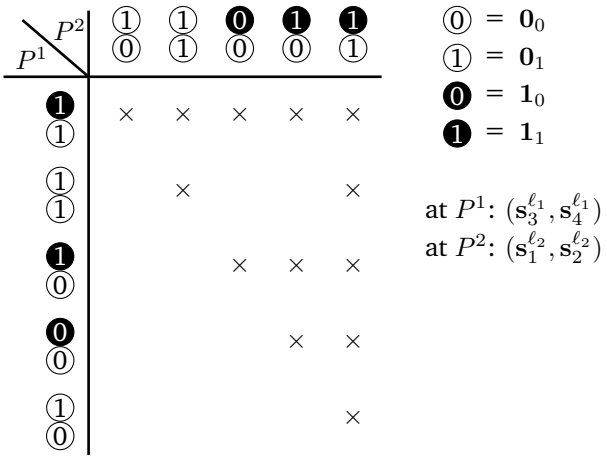


Fig. 7.8.: A matrix depicting the possible state transitions between two consecutive path gadgets. The rows are labeled with $(\mathbf{s}_3^{\ell_1}, \mathbf{s}_4^{\ell_1})$, $\ell_1 \in [5]$, and the columns are labeled with $(\mathbf{s}_1^{\ell_2}, \mathbf{s}_2^{\ell_2})$, $\ell_2 \in [5]$. An $\times$ marks possible state transitions in a connected dominating set.

Lemma 7.3.4. *If X satisfies $|X \cap P^1| \leq 14$ and $|X \cap P^2| \leq 14$, then $\mathbf{state}(X \cap P^1) = \mathbf{s}^{\ell_1}$ and $\mathbf{state}(X \cap P^2) = \mathbf{s}^{\ell_2}$ with $\ell_1 \leq \ell_2$.*

Additionally, for each $\ell \in [5]$, the set $X^\ell = X^\ell_{P^1} \cup X^\ell_{P^2}$ dominates or root-connects all inner join vertices, i.e., $\mathbf{state}_{X^\ell}(\{u^1_{2,1}, u^1_{2,2}, u^2_{1,1}, u^2_{1,2}\}) \subseteq \{\mathbf{0}_1, \mathbf{1}_1\}$.

Proof. By Lemma 7.3.2, there are $\ell_1, \ell_2 \in [5]$ such that $\mathbf{state}(X \cap P^1) = \mathbf{s}^{\ell_1}$ and $\mathbf{state}(X \cap P^2) = \mathbf{s}^{\ell_2}$. It remains to verify that $\ell_1 \leq \ell_2$. The main idea is to consider the states at the inner join vertices $u^1_{2,1}, u^1_{2,2}$, and $u^2_{1,1}, u^2_{1,2}$ and notice that at least one of these four vertices is not dominated or not root-connected whenever $\ell_1 > \ell_2$. Recall that these inner join vertices are only adjacent to vertices inside $P^1 + P^2$ and $N(\{u^1_{2,1}, u^1_{2,2}\}) \cap V(P^2) = \{u^2_{1,1}, u^2_{1,2}\}$ and $N(\{u^2_{1,1}, u^2_{1,2}\}) \cap V(P^1) = \{u^1_{2,1}, u^1_{2,2}\}$. Figure 7.8 depicts the possible state transitions.

First, if $\ell_1 \in \{3, 4, 5\}$ and $\ell_2 \in [2]$, then we have that $\mathbf{state}_{X \cap P^1}(u^1_{2,2}) = \mathbf{0}_0$ and $\mathbf{state}_{X \cap P^2}(\{u^2_{1,1}, u^2_{1,2}\}) \subseteq \{\mathbf{0}_0, \mathbf{0}_1\}$. Therefore, the vertex $u^1_{2,2}$ cannot be dominated by X in this case. Secondly, if $\ell_1 \in \{4, 5\}$ and $\ell_2 = 3$, then $\mathbf{1}_1 \notin \mathbf{state}_{X \cap P^1}(\{u^1_{2,1}, u^1_{2,2}\})$ and $\mathbf{state}_{X \cap P^2}(u^2_{1,1}) = \mathbf{1}_0$. Therefore, the vertex $u^2_{1,1}$ cannot be root-connected in X in this case. Lastly, if $(\ell_1, \ell_2) \in \{(2, 1), (5, 4)\}$, then we have $\mathbf{state}_{X \cap P^1}(\{u^1_{2,1}, u^1_{2,2}\}) \subseteq \{\mathbf{0}_0, \mathbf{0}_1\}$ and $\mathbf{state}_{X \cap P^2}(u^2_{1,2}) = \mathbf{0}_0$. Therefore, the vertex $u^2_{1,2}$ cannot be dominated by X in this case.

For the second part, fix some $\ell \in [5]$. By Lemma 7.3.3, we have that $\mathbf{state}(X^\ell \cap P^1) = \mathbf{state}(X^\ell \cap P^2) = \mathbf{s}^\ell$. We distinguish based on $\ell \in [5]$; also see Figure 7.9:

- $\ell = 1$: $u^2_{1,2}$ is dominated by $u^1_{2,1}$.
- $\ell = 2$: all four vertices are already dominated by Lemma 7.3.3.
- $\ell = 3$: $u^1_{2,1}$ root-connects $u^2_{1,1}$ and dominates $u^2_{1,2}$; $u^2_{1,1}$ dominates $u^1_{2,2}$.
- $\ell = 4$: reverse situation of $\ell = 3$.
- $\ell = 5$: reverse situation of $\ell = 1$.

This finishes the proof of the second part. □

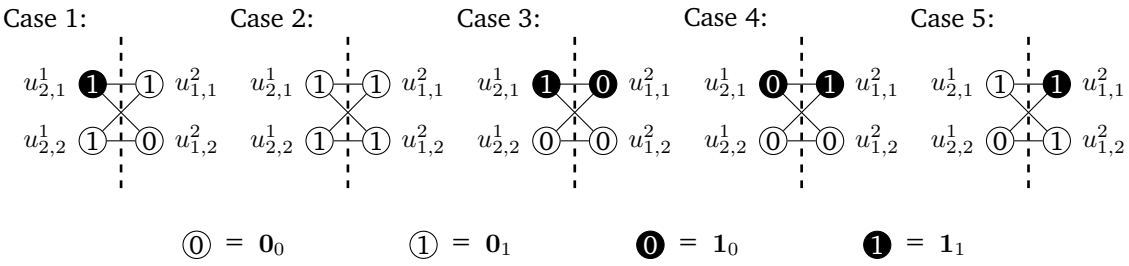


Fig. 7.9.: Case distinction to show $\mathbf{state}_{X^\ell}(\{u^1_{2,1}, u^1_{2,2}, u^2_{1,1}, u^2_{1,2}\}) \subseteq \{\mathbf{0}_1, \mathbf{1}_1\}$.

We say that a *cheat occurs* if $\ell_1 < \ell_2$. Creating arbitrarily long paths of the path gadgets P, Lemma 7.2.4 shows that at most $|\mathbf{States}| - 1 = 4 = \mathcal{O}(1)$ cheats occur on such a path which allows us to find a *cheat-free region* as outlined previously.

7.3.2 Complete Construction

Setup. Assume that Connected Dominating Set can be solved in time $\mathcal{O}^*((5-\varepsilon)^k)$ for some $\varepsilon > 0$ when given a linear k-expression. Given a Satisfiability-instance σ with n variables and m clauses, we construct an equivalent Connected Dominating Set instance with linear-clique-width approximately $n \log_5(2)$ so that the existence of such an algorithm for Connected Dominating Set would imply that the SETH is false.

We pick an integer β only depending on ε; the precise choice of β will be discussed at a later point. The variables of σ are partitioned into groups of size at most β, resulting in $t = \lceil n/\beta \rceil$ groups. Furthermore, we pick the smallest integer p that satisfies $5^p \geq 2^\beta$, i.e., $p = \lceil \log_5(2^\beta) \rceil$. We now begin with the construction of the Connected Dominating Set instance $(G = G(\sigma, \beta), \overline{b})$.

We create the root vertex $\hat{r}$ and attach a leaf $\hat{r}'$ which forces $\hat{r}$ into any connected dominating set. For every group $i \in [t]$, we create p long path-like gadgets $P^{i,j}$, $j \in [p]$, where each $P^{i,j}$ consists of $m(4tp+1)$ copies $P^{i,j,\ell}$, $\ell \in [m(4tp+1)]$, of the path gadget P and consecutive copies are connected by a join. More precisely, the vertices in some $P^{i,j,\ell}$ inherit their names from the generic path gadget P and the superscript of $P^{i,j,\ell}$ and for every $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)-1]$, the vertices $\{u^{i,j,\ell}_{2,1}, u^{i,j,\ell}_{2,2}\}$ are joined to $\{u^{i,j,\ell+1}_{1,1}, u^{i,j,\ell+1}_{1,2}\}$. The ends of each path $P^{i,j}$, i.e., the vertices $u^{i,j,1}_{1,1}, u^{i,j,1}_{1,2}, u^{i,j,m(4tp+1)}_{2,1}, u^{i,j,m(4tp+1)}_{2,2}$ are made adjacent to the root $\hat{r}$.

For every group $i \in [t]$ and column $\ell \in [m(4tp+1)]$, we create a *decoding gadget* $D^{i,\ell}$ in the same style as Cygan et al. [49] for Connected Vertex Cover[pathwidth]. Every variable group i has at most 2^β possible truth assignments and by choice of p we have that $5^p \geq 2^\beta$, so we can find an injective mapping $\kappa\colon \{0,1\}^\beta \to [5]^p$ which assigns to each truth

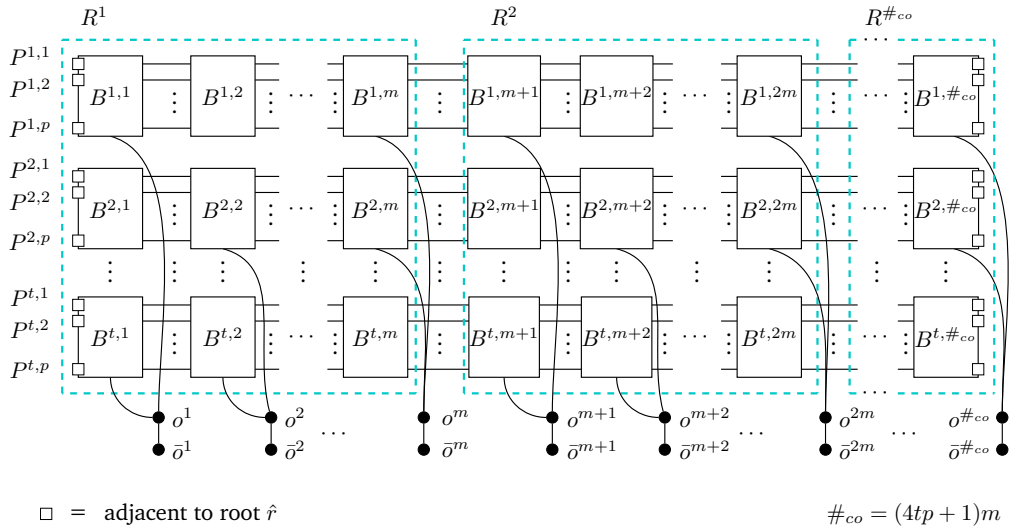


Fig. 7.10.: High-level construction for CONNECTED DOMINATING SET[linear-clique-width]. Every edge between two blocks $B^{i,\ell}$ and $B^{i,\ell+1}$ represents a join. The regions R^γ, highlighted by the dashed cyan rectangles, are important for the proof of Lemma 7.3.6.

assignment $\tau \in \{0,1\}^\beta$ a sequence $\kappa(\tau) \in [5]^p$. For each sequence $\mathbf{h} = (h_1, \ldots, h_p) \in [5]^p$, we create vertices $x^{i,\ell}_{\mathbf{h}}$, $\bar{x}^{i,\ell}_{\mathbf{h}}$, $y^{i,\ell}_{\mathbf{h}}$ and edges $\{x^{i,\ell}_{\mathbf{h}}, \bar{x}^{i,\ell}_{\mathbf{h}}\}$, $\{x^{i,\ell}_{\mathbf{h}}, y^{i,\ell}_{\mathbf{h}}\}$, $\{y^{i,\ell}_{\mathbf{h}}, \hat{r}\}$. We add the edge $\{x^{i,\ell}_{\mathbf{h}}, v^{i,j,\ell}_{h_j}\}$ for all $\mathbf{h} = (h_1, \ldots, h_p) \in [5]^p$ and $j \in [p]$. Finally, we create two adjacent vertices $z^{i,\ell}$ and $\bar{z}^{i,\ell}$ and edges $\{z^{i,\ell}, y^{i,\ell}_{\mathbf{h}}\}$ for all $\mathbf{h} \in [5]^p$. Each decoding gadget $D^{i,\ell}$ together with the adjacent path gadgets $P^{i,j,\ell}$, $j \in [p]$, forms the *block* $B^{i,\ell}$, see Figure 7.10 for a high-level depiction of the connections between different blocks.

Lastly, we construct the *clause gadgets*. We number the clauses of σ by $C_0, \ldots, C_{m-1}$. For every column $\ell \in [m(4tp+1)]$, we create an adjacent pair of vertices o^ℓ and $\bar{o}^\ell$. Let $\ell' \in [0, m-1]$ be the remainder of $(\ell - 1)$ modulo m; for all $i \in [t]$, $\mathbf{h} \in \kappa(\{0,1\}^\beta)$ such that $\kappa^{-1}(\mathbf{h})$ is a truth assignment for variable group i satisfying clause $C_{\ell'}$, we add the edge $\{o^\ell, y^{i,\ell}_{\mathbf{h}}\}$. See Figure 7.11 for a depiction of the decoding and clause gadget and the connections to the path gadgets.

Lemma 7.3.5. *If σ is satisfiable, then there exists a connected dominating set X of $G = G(\sigma, \beta)$ of size $|X| \leq (14tp + (5^p + 2)t + 1)m(4tp+1) + 1 = \overline{b}$.*

Proof. Let τ be a satisfying truth assignment of σ and let τ^i denote the restriction of τ to the i-th variable group for every $i \in [t]$ and let $\kappa(\tau^i) = \mathbf{h}^i = (h^i_1, \ldots, h^i_p) \in [5]^p$ be the corresponding sequence. The connected dominating set is given by

$$X = \{\hat{r}\} \cup \bigcup_{\ell \in [m(4tp+1)]} \left(\{o^\ell\} \cup \bigcup_{i \in [t]} \left(\{y^{i,\ell}_{\mathbf{h}^i}, z^{i,\ell}\} \cup \bigcup_{\mathbf{h} \in [5]^p} \{x^{i,\ell}_{\mathbf{h}}\} \cup \bigcup_{j \in [p]} X^{h^i_j}_{P^{i,j,\ell}} \right) \right),$$

where $X^{h^i_j}_{P^{i,j,\ell}}$ refers to the sets given by Lemma 7.3.3.

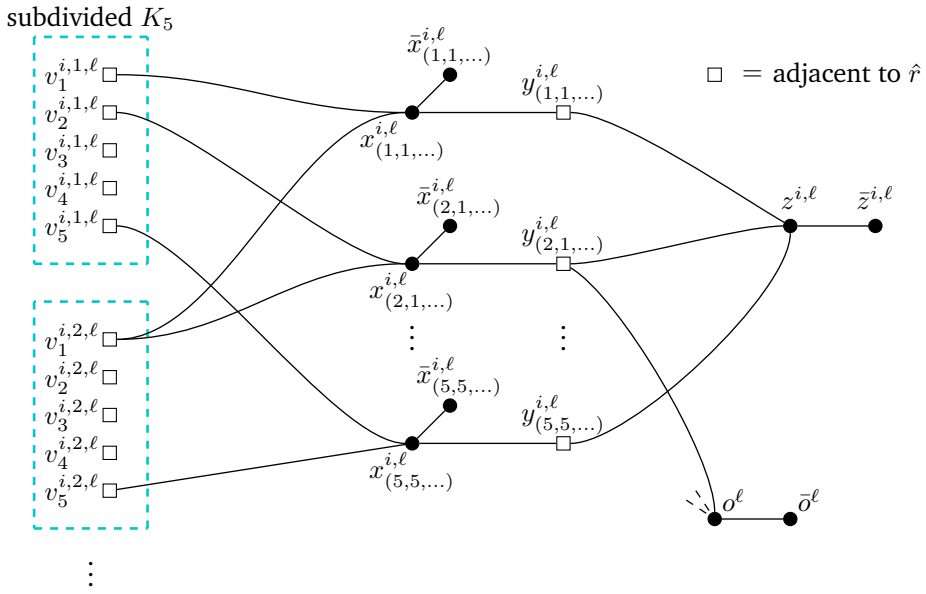


Fig. 7.11.: Decoding and clause gadget for CONNECTED DOMINATING SET[linear-clique-width].

Clearly, $|X| = \bar{b}$ as $|X^{h_j^i}_{P^{i,j,\ell}}| = 14$ for all i, j, ℓ, so it remains to prove that X is a connected dominating set. We begin by considering the path gadgets. First, notice that we have

$$\bigcup_{i\in[t]} \bigcup_{j\in[p]} \{u_{1,1}^{i,j,1}, u_{1,2}^{i,j,1}, u_{2,1}^{i,j,m(4tp+1)}, u_{2,2}^{i,j,m(4tp+1)}\} \subseteq N(\hat{r}),$$

hence the vertices at the ends of the paths $P^{i,j}$ are dominated or root-connected in X. Next, we invoke Lemma 7.3.3 and the second part of Lemma 7.3.4 to see that all vertices on the path gadgets are root-connected in X or dominated.

Fix $i \in [t]$, $\ell \in [m(4tp+1)]$, and consider the corresponding decoding gadget. Since $z^{i,\ell} \in X$ and $x^{i,\ell}_{\mathbf{h}} \in X$ for all $\mathbf{h} \in [5]^p$, all vertices in the decoding gadget are dominated by X. Furthermore, since $o^\ell \in X$, all vertices inside the clause gadget are dominated by X. Hence, X has to be a dominating set of G.

It remains to prove that the vertices in the decoding and clause gadgets that belong to X are also root-connected. Again, fix $i \in [t]$, $\ell \in [m(4tp+1)]$, and $\mathbf{h} = (h_1, \ldots, h_p) \in [5]^p \setminus \{\mathbf{h}^i\}$. Since $\mathbf{h} \neq \mathbf{h}^i$, there is some $j \in [p]$ such that $h_j \neq h_j^i$ and hence $v_{h_j}^{i,j,\ell} \in X$ by Lemma 7.3.3 which connects $x^{i,\ell}_{\mathbf{h}}$ to the root $\hat{r}$. The vertices $x^{i,\ell}_{\mathbf{h}^i}$ and $z^{i,\ell}$ are root-connected via $y^{i,\ell}_{\mathbf{h}^i} \in X$.

We conclude by showing that o^ℓ is root-connected for all $\ell \in [m(4tp+1)]$. Since τ is a satisfying truth assignment of σ, there is some variable group $i \in [t]$ such that τ^i already satisfies clause $C_{\ell'}$, where ℓ' is the remainder of $(\ell - 1)$ modulo m. By construction of G and X, the vertex $y^{i,\ell}_{\mathbf{h}^i} \in X$ is adjacent to o^ℓ, since $\kappa(\tau^i) = \mathbf{h}^i$, and connects o^ℓ to the root $\hat{r}$. This shows that all vertices of X are root-connected, so $G[X]$ has to be connected. □

Lemma 7.3.6. *If there exists a connected dominating set X of $G = G(\sigma, \beta)$ of size $|X| \leq (14tp + (5^p + 2)t + 1)m(4tp+1) + 1 = \bar{b}$, then σ is satisfiable.*

Proof. We begin by arguing that X has to satisfy $|X| = \bar{b}$. First, we must have that $\hat{r} \in X$, because $\hat{r}$ has a neighbor of degree 1. By Lemma 7.3.1, we have that $|X \cap P^{i,j,\ell}| \geq 14$ for all $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$. In every decoding gadget, i.e., one for every $i \in [t]$ and $\ell \in [m(4tp+1)]$, the set $\{z^{i,\ell}\} \cup \bigcup_{\mathbf{h} \in [5]^p} x^{i,\ell}_{\mathbf{h}}$ has to be contained in X, since every vertex in this set has a neighbor of degree 1. Furthermore, to connect $z^{i,\ell}$ to $\hat{r}$, at least one of the vertices $y^{i,\ell}_{\mathbf{h}}$, $\mathbf{h} \in [5]^p$, has to be contained in X. Hence, X must contain at least $5^p + 2$ vertices per decoding gadget. Lastly, $o^\ell \in X$ for all $\ell \in [m(4tp+1)]$, since o^ℓ has a neighbor of degree 1. Since we have only considered disjoint vertex sets, this shows that $|X| = \bar{b}$ and all of the previous inequalities have to be tight.

By Lemma 7.3.2, we know that X assumes one of the five possible states on each $P^{i,j,\ell}$. Fix some $P^{i,j} = \bigcup_{\ell \in [m(4tp+1)]} P^{i,j,\ell}$ and note that due to Lemma 7.3.4 the state can change at most four times along $P^{i,j}$. Such a state change is called a *cheat*. Let $\gamma \in [0, 4tp]$ and define the γ-th *region* $R^\gamma = \bigcup_{i \in [t]} \bigcup_{j \in [p]} \bigcup_{\ell=\gamma m+1}^{(\gamma+1)m} P^{i,j,\ell}$. Since there are $4tp+1$ regions, there is at least one region R^γ such that no cheat occurs in R^γ. We will consider this region for the remainder of the proof and read off a satisfying truth assignment from this region.

For $i \in [t]$, define $\mathbf{h}^i = (h^i_1, \ldots, h^i_p) \in [5]^p$ such that $v^{i,j,\gamma m+1}_{h^i_j} \notin X$ for all $j \in [p]$; this is well-defined by Lemma 7.3.2. Since R^γ does not contain any cheats, the definition of $\mathbf{h}^i$ is independent of which column $\ell \in [\gamma m+1, (\gamma+1)m]$ we consider. For every $i \in [t]$ and $\ell \in [\gamma m+1, (\gamma+1)m]$, we claim that $y^{i,\ell}_{\mathbf{h}} \in X$ if and only if $\mathbf{h} = \mathbf{h}^i$. We have already established that for every i and ℓ, there is exactly one $\mathbf{h}$ such that $y^{i,\ell}_{\mathbf{h}} \in X$. Consider the vertex $x^{i,\ell}_{\mathbf{h}^i} \in X$, its neighbors in G are $v^{i,1,\ell}_{h^i_1}, v^{i,2,\ell}_{h^i_2}, \ldots, v^{i,p,\ell}_{h^i_p}$, $\bar{x}^{i,\ell}_{\mathbf{h}^i}$, and $y^{i,\ell}_{\mathbf{h}^i}$. By construction of $\mathbf{h}^i$ and the tight allocation of the budget, we have $(N_G(x^{i,\ell}_{\mathbf{h}^i}) \setminus \{y^{i,\ell}_{\mathbf{h}^i}\}) \cap X = \emptyset$. Therefore, X has to include $y^{i,\ell}_{\mathbf{h}^i}$ to connect $x^{i,\ell}_{\mathbf{h}^i}$ to the root $\hat{r}$. This shows the claim.

For $i \in [t]$, we define the truth assignment τ^i for variable group i by taking an arbitrary truth assignment if $\mathbf{h}^i \notin \kappa(\{0,1\}^\beta)$ and setting $\tau^i = \kappa^{-1}(\mathbf{h}^i)$ otherwise. By setting $\tau = \bigcup_{i \in [t]} \tau^i$ we obtain a truth assignment for all variables and we claim that τ satisfies σ. Consider some clause $C_{\ell'}$, $\ell' \in [0, m-1]$, and let $\ell = \gamma m + \ell' + 1$. We have already argued that $o^\ell \in X$ and to connect o^ℓ to the root $\hat{r}$, there has to be some $y^{i,\ell}_{\mathbf{h}} \in N_G(o^\ell) \cap X$. By the previous claim, $\mathbf{h} = \mathbf{h}^i$ for some $i \in [t]$ and therefore τ^i, and also τ, satisfy clause $C_{\ell'}$ due to the construction of G. Because the choice of $C_{\ell'}$ was arbitrary, τ has to be a satisfying assignment of σ. □

Lemma 7.3.7. *The constructed graph $G = G(\sigma, \beta)$ has* $\text{lin-cw}(G) \leq tp + 3 \cdot 5^p + \mathcal{O}(1)$ *and a linear clique-expression of this width can be constructed in polynomial time.*

Proof. We will describe how to construct a linear clique-expression for G of width $tp + 3 \cdot 5^p + \mathcal{O}(1)$. The clique-expression will use one path label (i, j) for every long path $P^{i,j}$, $i \in [t]$, $j \in [p]$, temporary decoding labels for every vertex of a decoding gadget, i.e. $3 \cdot 5^p + 2$ many, temporary gadget labels for every vertex of a path gadget, two temporary clause labels for the clause gadget, one label for the root vertex and a trash label.

The construction of the clique-expression is described in Algorithm 4 and the idea is to proceed column by column and group by group in each column. By reusing the temporary labels, we keep the total number of labels small. The maximum number of labels used simultaneously occurs in line 7 and is $tp + (3 \cdot 5^p + 2) + \mathcal{O}(1)$. This concludes the proof. □

Algorithm 4: Constructing a linear clique-expression for G.

Introduce $\hat{r}$ with root label and $\hat{r}'$ with trash label and add the edge $\{\hat{r}, \hat{r}'\}$;
for $\ell \in [m(4tp+1)]$ **do**
  Introduce o^ℓ and $\bar{o}^\ell$ and the edge $\{o^\ell, \bar{o}^\ell\}$ with the clause labels;
  for $i \in [t]$ **do**
    Build $D^{i,\ell}$ using decoding labels and add edges to o^ℓ;
    for $j \in [p]$ **do**
      Build $P^{i,j,\ell}$ using gadget labels and add edges to $D^{i,j}$;
      Join $u^{i,j,\ell}_{1,1}$ and $u^{i,j,\ell}_{1,2}$ to path label (i,j) if $\ell > 1$ and to root label otherwise;
      Relabel path label (i,j) to trash label;
      Relabel $u^{i,j,\ell}_{2,1}$ and $u^{i,j,\ell}_{2,2}$ to path label (i,j);
      Relabel all other gadget labels to the trash label;
    Relabel all decoding labels to the trash label;
  Relabel all clause labels to the trash label;

Theorem 7.3.8. *There is no algorithm that solves* Connected Dominating Set*, given a linear k-expression, in time $\mathcal{O}^*((5-\varepsilon)^k)$ for some $\varepsilon > 0$, unless CNF-SETH fails.*

Proof. Assume that there exists an algorithm $\mathbb{A}$ that solves Connected Dominating Set in time $\mathcal{O}^*((5-\varepsilon)^k)$ for some $\varepsilon > 0$ given a linear k-expression. Given β, we define $\delta_1 < 1$ such that $(5-\varepsilon)^{\log_5(2)} = 2^{\delta_1}$ and δ_2 such that $(5-\varepsilon)^{1/\beta} = 2^{\delta_2}$. By picking β large enough, we can ensure that $\delta = \delta_1 + \delta_2 < 1$. We will show how to solve Satisfiability using $\mathbb{A}$ in time $\mathcal{O}^*(2^{\delta n}\operatorname{poly}(m)) = \mathcal{O}^*(2^{\delta n})$, where n is the number of variables, thus contradicting CNF-SETH, Conjecture 2.1.1.

Given a Satisfiability instance σ, we construct $G = G(\sigma, \beta)$ and the linear clique-expression from Lemma 7.3.7 in polynomial time, note that we have $\beta = \mathcal{O}(1)$ and hence $p = \mathcal{O}(1)$; recall $p = \lceil \log_5(2^\beta) \rceil$. We then run $\mathbb{A}$ on G and return its answer. This is correct by Lemma 7.3.5 and Lemma 7.3.6. Due to Lemma 7.3.7, the running time is

$$\begin{aligned}
& \mathcal{O}^*\left((5-\varepsilon)^{tp+3\cdot 5^p+\mathcal{O}(1)}\right) \leq \mathcal{O}^*\left((5-\varepsilon)^{tp}\right) && \leq \mathcal{O}^*\left((5-\varepsilon)^{\lceil \frac{n}{\beta} \rceil p}\right) \\
\leq\ & \mathcal{O}^*\left((5-\varepsilon)^{\frac{n}{\beta}p}\right) && \leq \mathcal{O}^*\left((5-\varepsilon)^{\frac{n}{\beta}\lceil \log_5(2^\beta)\rceil}\right) && \leq \mathcal{O}^*\left((5-\varepsilon)^{\frac{n}{\beta}\log_5(2^\beta)}(5-\varepsilon)^{\frac{n}{\beta}}\right) \\
\leq\ & \mathcal{O}^*\left(2^{\delta_1 \beta \frac{n}{\beta}} 2^{\delta_2 n}\right) && \leq \mathcal{O}^*\left(2^{(\delta_1+\delta_2)n}\right) && \leq \mathcal{O}^*\left(2^{\delta n}\right),
\end{aligned}$$

hence completing the proof. □

8 Lower Bounds Parameterized By Modular-Treewidth

8.1 General Approach

In this section, we prove the tight lower bounds for Connected Vertex Cover and Feedback Vertex Set parameterized by modular-treewidth. We do not need the full power of the modular decomposition, twinclass-variants are already sufficient. Since we follow the high-level construction principle of Lokshtanov et al. [126] again, the lower bounds already apply to the linear version of our width parameter. Therefore, we prove tight lower bounds for Connected Vertex Cover[twinclass-pathwidth] and Feedback Vertex Set[twinclass-pathwidth], which transfer to modular-treewidth as $\text{mod-tw} \preceq \text{mod-pw} \preceq \text{tc-pw}$ by Lemma 2.4.22.

Connections Between Path Gadgets. For twinclass-pathwidth, two consecutive path gadgets in the same row are connected via a single twinclass, i.e., such twinclass acts as a vertex separator of the row; in this way, each row of the construction contributes one unit of twinclass-pathwidth. Whereas for cutwidth and clique-width adjacent path gadgets are connected by some type of edge separator, for pathwidth and twinclass-pathwidth we have vertex separators, which has some nontrivial consequences for our design approach. As we consider vertex selection problems, the solutions might contain vertices of these separators. Since we need to balance the costs of the different path gadget states, where the contained separator vertices can depend on the state, we also have to account for the cost due to the separator vertices. To handle this, we consider the separators to be part of the path gadgets now, as opposed to before. The consequence is that the path gadgets will be *asymmetric* in some sense: the left boundary consists of the separating twinclass and the right boundary consists of possibly multiple twinclasses/vertices that will be connected to the left boundary of the next path gadget. Therefore, also the states communicated to the boundaries of the path gadget are of different kinds, where we have more flexibility at the right boundary by using multiple twinclasses.

Determining a Transition Order. Between two twinclasses, either no edges exist or all edges exist, i.e., we have a join between them in the latter case. Therefore, it does make sense to first study what happens across a join, which we can simply copy from the clique-width investigation for the case of Connected Vertex Cover. However, similar to the algorithms for parameterization by modular-treewidth, we see that the connectivity properties of vertices inside a twinclass behave in a *homogeneous* way, which causes us to lose some states, at least at the left boundary, that were applicable for clique-width. For both

problems, the remaining states are not sufficient to prove a tight lower bound, instead, we have to make use of the allowed asymmetry of the path gadget. By using several vertices that are not in the same twinclass at the right boundary, we can essentially recover some of the lost states and obtain a compatibility matrix with a sufficient amount of structure allowing us to prove the tight lower bounds.

Designing the Path Gadget. We use the same approach as for clique-width, i.e., the path gadget has a *clique-like center* and the two boundary parts. However, the design of the left boundary and the right boundary is not necessarily as similar as before due to the asymmetry. In particular, the design of the left boundary, i.e., the separating twinclass, is challenging, as previously we could attach gadgets to singular boundary vertices, but now we can only attach gadgets to the twinclass as a whole. Hence, the clique-width approach of isolating properties of the boundary vertices via indicator pairs does not directly apply. Additionally, this also complicates the balancing of the costs of the various path gadget states.

Decoding and Clause Gadgets. Due to the clique-like center of the path gadgets, we can again reuse already-known constructions for the decoding and clause gadget. For CONNECTED VERTEX COVER, we use the construction of the CONNECTED VERTEX COVER[pathwidth] lower bound by Cygan et al. [50]. For FEEDBACK VERTEX SET, we use a simplified construction compared to the FEEDBACK VERTEX SET[pathwidth] lower bound of Cygan et al.[50], which we will reuse in Part III when proving lower bounds for modulator-parameters.

8.2 Connected Vertex Cover Lower Bound

This subsection is devoted to proving that CONNECTED VERTEX COVER[twinclass-pathwidth] cannot be solved in time $\mathcal{O}^*((5-\varepsilon)^{\text{tc-pw}(G)})$ for some $\varepsilon > 0$ unless the CNF-SETH, Conjecture 2.1.1, fails. We first design the path gadget and analyze it in isolation and afterwards we present the complete construction. The decoding gadgets are directly adapted from the lower bound for CONNECTED VERTEX COVER[pathwidth] given by Cygan et al. [50].

8.2.1 Path Gadget Construction and Analysis

Root. We create a vertex $\hat{r}$ called the *root* and attach a vertex $\hat{r}'$ of degree 1 to ensure that every connected vertex cover contains $\hat{r}$. Given a subset $X \subseteq V(G)$ with $\hat{r} \in X$, a vertex $v \in X$ is *root-connected* in X if there is a $v, \hat{r}$-path in $G[X]$. We just say *root-connected* if X is clear from the context. Note that $G[X]$ is connected if and only if all vertices of X are root-connected in X.

Determining a Transition Order. We first follow the state analysis of the clique-width case. The possible *vertex states* with respect to a partial solution $X \subseteq V(G)$ are $\mathbf{0}$, $\mathbf{1}_0$, $\mathbf{1}_1$, representing that a vertex is not in X (state $\mathbf{0}$), in X and root-connected (state $\mathbf{1}_1$) or not (state $\mathbf{1}_0$). As for labels in the clique-width case, we can describe the state of a twinclass by listing which distinct vertex appear in the twinclass, thus the possible *twinclass states* are

X_1 vs. X_2	$\{\mathbf{0}\}$	$\{\mathbf{1}_0\}$	$\{\mathbf{1}_1\}$	$\{\mathbf{1}_0,\mathbf{0}\}$	$\{\mathbf{1}_1,\mathbf{0}\}$
$\{\mathbf{0}\}$	0	0	1	0	0
$\{\mathbf{1}_0\}$	0	0	1	0	1
$\{\mathbf{1}_1\}$	1	1	1	1	1
$\{\mathbf{1}_0,\mathbf{0}\}$	0	0	1	0	0
$\{\mathbf{1}_1,\mathbf{0}\}$	0	1	1	0	0

Tab. 8.1.: The compatibility matrix for CONNECTED VERTEX COVER with respect to a join between two twinclasses.

X_1 vs. X_2	$\{\mathbf{1}_0,\mathbf{0}\}$	$\{\mathbf{1}_1,\mathbf{0}\}$	$\{\mathbf{1}_0\}$	$\{\mathbf{1}_1,\mathbf{1}_0\}$	$\{\mathbf{1}_1\}$
$\{\mathbf{1}_1\}$	1	1	1	1	1
$\{\mathbf{1}_0\}$	0	1	0	1	1
$\{\mathbf{1}_1,\mathbf{0}\}$	0	0	1	1	1
$\{\mathbf{1}_0,\mathbf{0}\}$	0	0	0	1	1
$\{\mathbf{0}\}$	0	0	0	0	1

Tab. 8.2.: The triangular compatibility matrix for CONNECTED VERTEX COVER[twinclass-pathwidth] obtained by using asymmetric states.

described by the power set $\mathcal{P}(\{\mathbf{0},\mathbf{1}_0,\mathbf{1}_1\})$. Considering a join between two twinclasses, we obtain the same compatibility matrix as for clique-width, see Table 7.1. However, in every twinclass C that avoids the root $\hat{r}$, the states $\mathbf{1}_0$ and $\mathbf{1}_1$ cannot simultaneously occur: if $v \in C \cap X$ is root-connected in $G[X]$, then we also obtain a path to the root $\hat{r}$ for any other vertex $w \neq v \in C \cap X$ because every twinclass is a module. Therefore, only the compatibility matrix in Table 8.1 remains. However, this compatibility matrix has only rank 4 and can therefore not be rearranged to a triangular matrix of size 5×5.

Introducing Asymmetry. To obtain a compatibility matrix containing a large enough triangular submatrix, we make use of the allowed asymmetry for path gadgets in this case. Instead of considering a single twinclass at the right boundary, we consider multiple vertices that do not belong to the same twinclass. As all vertices at the right boundary will be adjacent to the left boundary of the next path gadget, we may consider the right boundary as a single label and thus recover the heterogeneous states $\{\mathbf{1}_0,\mathbf{1}_1\}$ and $\{\mathbf{0},\mathbf{1}_0,\mathbf{1}_1\}$. In the compatibility matrix for the clique-width case, Table 7.1, this allows us to build a triangular submatrix by picking rows corresponding to homogeneous states and columns corresponding to homogeneous and heterogeneous states (or swapping the role of rows and columns). Doing so, we obtain the triangular matrix in Table 8.2.

Formal Definition of States. We collect the three vertex states, also called *atomic states*, in the set $\mathbf{Atoms} = \{\mathbf{0},\mathbf{1}_0,\mathbf{1}_1\}$ and define the two predicates $\texttt{sol},\texttt{conn}\colon \mathbf{Atoms} \to \{0,1\}$ by $\texttt{sol}(\mathbf{a}) = [\mathbf{a} \in \{\mathbf{1}_0,\mathbf{1}_1\}]$ and $\texttt{conn}(\mathbf{a}) = [\mathbf{a} = \mathbf{1}_1]$. The meaning of these symbols is the same as before. As the twinclass and label states used in Table 8.2 can be attained by just using two vertices each, we obtain the following five *path gadget states*, each consisting of four atomic states, by pairing states along the main diagonal:

- $\mathbf{s}^1 = (\mathbf{0}\ ,\mathbf{0}\ ,\mathbf{1}_1,\mathbf{1}_1)$,
- $\mathbf{s}^2 = (\mathbf{1}_0,\mathbf{0}\ ,\mathbf{1}_1,\mathbf{1}_0)$,
- $\mathbf{s}^3 = (\mathbf{1}_1,\mathbf{0}\ ,\mathbf{1}_0,\mathbf{1}_0)$,
- $\mathbf{s}^4 = (\mathbf{1}_0,\mathbf{1}_0,\mathbf{1}_1,\mathbf{0}\ )$,
- $\mathbf{s}^5 = (\mathbf{1}_1,\mathbf{1}_1,\mathbf{1}_0,\mathbf{0}\ )$.

The states are numbered in the transition order following from Table 8.2. We collect the five states in the set $\mathbf{States} = \{\mathbf{s}^1, \ldots, \mathbf{s}^5\}$ and use the notation $\mathbf{s}^\ell_i \in \mathbf{Atoms}$, $i \in [4]$, $\ell \in [5]$, to refer to the i-th atomic component of state $\mathbf{s}^\ell$.

Given a partial solution $Y \subseteq V(G)$, we formally associate to each vertex its state with respect to Y via the map $\mathbf{state}_Y \colon V(G) \to \mathbf{Atoms}$ by

$$\mathbf{state}_Y(v) = \begin{cases} \mathbf{0} & \text{if } v \notin Y, \\ \mathbf{1}_0 & \text{if } v \in Y \text{ and } v \text{ is not root-connected in } Y \cup \{\hat{r}\}, \\ \mathbf{1}_1 & \text{if } v \in Y \text{ and } v \text{ is root-connected in } Y \cup \{\hat{r}\}. \end{cases}$$

Path Gadget Construction. The path gadget P is constructed as follows. We create 15 *central* vertices $w_{\ell,i}$, $\ell \in [5]$, $i \in [3]$, in 5 sets $W_\ell = \{w_{\ell,1}, w_{\ell,2}, w_{\ell,3}\}$ of size 3 and each set will form a twinclass. We create 2 *input* vertices u_1, u_2, 4 *cost* vertices $w_{+,1}, \ldots, w_{+,4}$, 5 *clique* vertices $v_1, \ldots, v_5$, and 5 *complement* vertices $\bar{v}_1, \ldots, \bar{v}_5$. Furthermore, for every $f \in [4]$, we create 2 *auxiliary* vertices $a_{1,f}, a_{2,f}$, 2 *indicator* vertices $b_{0,f}, b_{1,f}$, and 2 *connectivity* vertices $c_{0,f}, c_{1,f}$. Finally, we create 4 further auxiliary vertices $\bar{a}_{1,1}, \bar{a}_{2,1}, \bar{a}_{1,2}, \bar{a}_{2,2}$ and 4 further connectivity vertices $\bar{c}_{0,1}, \bar{c}_{1,1}, \bar{c}_{0,2}, \bar{c}_{1,2}$. The vertices $a_{1,4}$ and $\bar{a}_{1,2}$ will also be called *output* vertices.

We add edges such that the central sets W_ℓ, $\ell \in [5]$, are pairwise adjacent twinclasses, i.e. they induce a complete 5-partite graph, and such that the clique vertices v_ℓ, $\ell \in [5]$, form a clique. Each complement vertex $\bar{v}_\ell$, $\ell \in [5]$, is made adjacent to W_ℓ and to v_ℓ. The cost vertices $w_{+,1}$ and $w_{+,2}$ are made adjacent to W_1; $w_{+,3}$ is made adjacent to W_2; and $w_{+,4}$ is made adjacent to W_3.

For every $f \in [4]$, we add edges $\{a_{1,f}, a_{2,f}\}$, $\{a_{2,f}, b_{1,f}\}$, $\{b_{1,f}, b_{0,f}\}$, $\{b_{0,f}, a_{1,f}\}$, forming a C_4, and the edges $\{a_{1,f}, c_{1,f}\}$ and $\{c_{0,f}, c_{1,f}\}$. For every $i \in [2]$, we add edges $\{\bar{a}_{1,i}, \bar{a}_{2,i}\}$, $\{\bar{a}_{1,i}, \bar{c}_{1,i}\}$, $\{\bar{c}_{0,i}, \bar{c}_{1,i}\}$. The input vertices u_1 and u_2 are made adjacent to each $a_{1,f}$ for $f \neq 4$ and they are made adjacent to $\bar{a}_{1,1}$.

All vertices except $\{a_{1,f} : f \in [4]\} \cup \{\bar{a}_{1,i}, \bar{a}_{2,i} : i \in [2]\} \cup \{u_1, u_2\}$ are made adjacent to the root $\hat{r}$. Finally, we describe how to connect the central vertices to the rest. Each twinclass W_ℓ, $\ell \in [5]$, is made adjacent to $b_{[\mathbf{s}^\ell_2 = \mathbf{0}],f}$ and to $c_{[\mathbf{s}^\ell_1 = \mathbf{s}^\ell_2],f}$ for all $f \in [4]$ and W_ℓ is also made adjacent to $\bar{c}_{[\mathbf{s}^\ell_1 \neq \mathbf{1}_0],1}$ and $\bar{c}_{[\mathbf{s}^\ell_1 \neq \mathbf{1}_1],2}$. The construction is depicted in Figure 8.1 and Figure 8.2.

We emphasize that the graphs $P[\{a_{1,f}, a_{2,f}, b_{0,f}, b_{1,f}, c_{0,f}, c_{1,f}\} \cup \bigcup_{\ell \in [5]} W_\ell]$, $f \in [4]$, are all isomorphic to each other, however the first three are also adjacent to the input vertices u_1 and u_2, whereas the fourth one is not. To study the path gadget P, we mostly consider the parts in Figure 8.1; the parts in Figure 8.2 are considerably simpler and will later allow us to simply attach the standard decoding gadget already used by Cygan et al. [50] for Connected Vertex Cover[pathwidth].

Behavior of Single Path Gadget. Having given the formal construction of the path gadget, we can start analyzing it. For this sake, we assume that G is a graph that contains $P + \hat{r}$ as an induced subgraph and that only the input vertices u_1, u_2, the output vertices $a_{1,4}, \bar{a}_{1,2}$, and the clique vertices $v_\ell, \ell \in [5]$, have neighbors outside this copy of $P + \hat{r}$. Furthermore, we assume that $\{u_1, u_2\}$ is a twinclass in G. Let X be a vertex cover of G with $\hat{r} \in X$. We

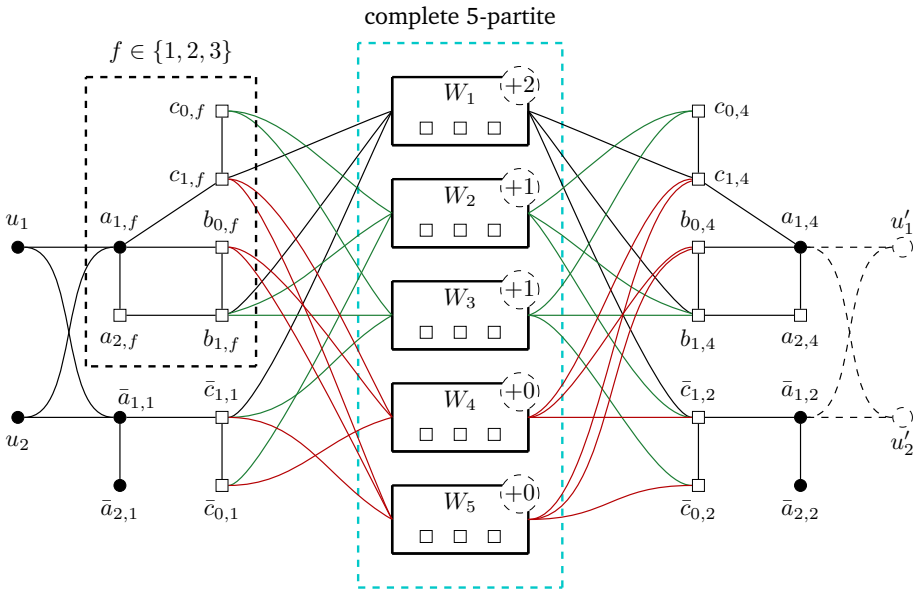


Fig. 8.1.: The main part of path gadget for Connected Vertex Cover[twinclass-pathwidth]. Vertices depicted with a rectangle are adjacent to the root vertex $\hat{r}$. The graph in the black dashed rectangle appears thrice with the same connections to the remaining vertices. The vertices inside the cyan dashed rectangle induce a complete 5-partite graph. The dashed circles at the central vertices indicate the number of cost vertices attached to this set, and the dashed vertices and edges at the right indicate how to connect to the next copy of the path gadget.

study the behavior of such vertex covers on P; we will abuse notation and write $X \cap P$ instead of $X \cap V(P)$.

Observe that the set

$$\begin{aligned} M = &\{\{a_{1,f}, a_{2,f}\}, \{b_{0,f}, b_{1,f}\}, \{c_{0,f}, c_{1,f}\} : f \in [4]\} \\ &\cup \{\{\bar{a}_{1,i}, \bar{a}_{2,i}\}, \{\bar{c}_{0,i}, \bar{c}_{1,i}\} : i \in [2]\} \\ &\cup \{\{v_\ell, \bar{v}_\ell\} : \ell \in [5]\} \end{aligned}$$

is a matching in P of size $4 \cdot 3 + 2 \cdot 2 + 5 = 21$. By also taking into account the behavior on the central clique-like part, we obtain the following lower bound for $|X \cap P|$. Additionally, by analyzing the behavior between the left boundary and the center more closely when $G[X]$ is connected, we see that the cost vertices balance the costs of the vertices inside the twinclass at the left boundary.

Lemma 8.2.1. *We have that* $|\{\ell \in [5] : W_\ell \subseteq X\}| \geq 4$ *and* $|X \cap P| \geq |M| + 4 \cdot 3 = 33$. *If* $G[X]$ *is connected, then* $|X \cap P| \geq |M| + 4 \cdot 3 + 2 = 35$ *and in case of equality,* $|X \cap \{u_1, u_2, w_{+,1}, \ldots, w_{+,4}\}| = 2$ *and there is a unique* $\ell \in [5]$ *such that* $W_\ell \not\subseteq X$.

Proof. The vertex set $\bigcup_{\ell \in [5]} W_\ell$ induces a complete 5-partite graph disjoint from the matching M. Any vertex cover must contain at least 4 of the 5 partition classes completely, otherwise there is an edge that is not covered, and since each class is of size 3, this accounts for $4 \cdot 3 = 12$ further vertices. This shows that $|X \cap P| \geq |M| + 4 \cdot 3 = 33$.

If X completely contains all W_ℓ, $\ell \in [5]$, then it immediately follows that $|X \cap P| \geq 36$, so if $|X \cap P| = 35$, then there is an unique $\ell \in [5]$ such that $W_\ell \not\subseteq X$. If $\ell = 1$, then we

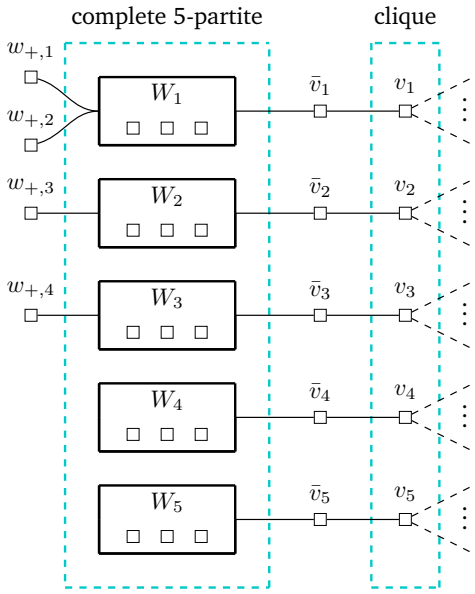


Fig. 8.2.: A depiction of the remaining parts of the path gadget P for CONNECTED VERTEX COVER[twinclass-pathwidth], which will be connected to the decoding gadget. All vertices that are depicted with a rectangle are adjacent to the root vertex $\hat{r}$. The vertices inside the cyan dashed rectangle induce a complete 5-partite graph or a clique respectively. Only the clique vertices have neighbors outside of P.

must have $w_{+,1}, w_{+,2} \in X$, so $|X \cap P| \geq 35$. Before we proceed with the remaining proof, notice that $A_f = \{a_{1,f}, a_{2,f}, b_{0,f}, b_{1,f}\}$ induces a C_4 for all $f \in [4]$, so if $|X \cap A_f| = 2$, then $X \cap A_f \in \{\{a_{1,f}, b_{1,f}\}, \{a_{2,f}, b_{0,f}\}\}$, i.e., X must pick an antipodal pair from A_f.

For the remainder of the proof, assume that $G[X]$ is connected. Suppose that $X \cap \{u_1, u_2\} = \emptyset$, then $a_{1,f} \in X$ for all $f \in [3]$ and $a_{1,f}$ must be root-connected in X. If $\ell \in \{2, 3\}$, then $b_{1,f}, c_{0,f} \in X$, so whichever neighbor of $a_{1,f}$ we choose for the sake of root-connectedness, the size of X increases by one for every $f \in [3]$. If $\ell \in \{4, 5\}$, then $b_{0,f} \in X$, so $a_{1,f}$ is root-connected, but we need to pick another vertex of A_f to cover the remaining edge induced by A_f, again increasing the size of X. In summary, we obtain $|X \cap P| \geq 36$ if $\ell > 1$ and $X \cap \{u_1, u_2\} = \emptyset$.

Suppose that $|X \cap \{u_1, u_2\}| = 1$ and without loss of generality $u_1 \in X$ and $u_2 \notin X$. Again, we must have $a_{1,f} \in X$ for all $f \in [3]$. If $\ell \in \{2, 3\}$, we have that $w_{+,3} \in X$ or $w_{+,4} \in X$. If $\ell \in \{4, 5\}$, we again see that $|X \cap A_f| \geq 3$ for all $f \in [3]$ and hence $|X \cap P| \geq 37$, so $|X \cap P| \geq 35$ in either case.

By the previous arguments, we see that $|X \cap P| = 35$ and $X \cap \{u_1, u_2\} = \emptyset$ implies that $\ell = 1$; $|X \cap P| = 35$ and $|X \cap \{u_1, u_2\}| = 1$ implies that $\ell \in \{2, 3\}$; $|X \cap P| = 35$ and $|X \cap \{u_1, u_2\}| = 2$ implies that $\ell \in \{4, 5\}$. So, the equation $|X \cap \{u_1, u_2, w_{+,1}, \ldots, w_{+,4}\}| = 2$ follows. □

Next, we want to show that the appropriate states, i.e., those given by $\mathbf{s}^1, \ldots, \mathbf{s}^5$, are propagated to the left and right boundary depending on which central twinclass is not fully contained in X. For any partial solution $Y \subseteq V(P)$, we define the 4-tuple

$$\mathbf{state}(Y) = (\mathbf{state}_Y(u_1), \mathbf{state}_Y(u_2), \mathbf{state}_Y(\bar{a}_{1,2}), \mathbf{state}_Y(a_{1,4})).$$

If $G[X]$ is connected, then the construction of P and the restrictions on how P connects to the remaining graph imply that any vertex $v \in V(P)$ with $\mathbf{state}_{X\cap P}(v) = \mathbf{1}_0$ must be root-connected in X via a path that leaves through the input vertices or output vertices.

We say that a vertex subset $Y \subseteq V(G)$ is *canonical* with respect to the twinclass $\{u_1, u_2\}$ if $u_2 \in Y$ implies $u_1 \in Y$; we will just say that Y is canonical if $\{u_1, u_2\}$ is clear from the context. Since $\{u_1, u_2\}$ is a twinclass, we can always assume that we are working with a canonical subset.

Lemma 8.2.2. *If X is canonical, $G[X]$ is connected, and $|X \cap P| \leq 35$, then $|X \cap P| = 35$ and there is an unique $\ell \in [5]$ such that $v_\ell \notin X$ and we have that $\mathbf{state}(X \cap P) = \mathbf{s}^\ell$.*

Proof. Lemma 8.2.1 implies that $|X \cap P| = 35$, $|X \cap \{u_1, u_2, w_{+,1}, \ldots, w_{+,4}\}| = 2$, that X contains exactly one endpoint of each edge in M and that there is an unique $\ell \in [5]$ such that $W_\ell \not\subseteq X$. To cover all edges between W_ℓ and $\bar{v}_\ell$, we must have that $\bar{v}_\ell \in X$ and $v_\ell \notin X$, since $\{\bar{v}_\ell, v_\ell\} \in M$. Furthermore, we must have $X \cap \{v_1, \ldots, v_5\} = \{v_1, \ldots, v_5\} \setminus \{v_\ell\}$, because else X does not cover the clique induced by $v_1, \ldots, v_5$. So, the uniqueness of v_ℓ follows.

Recall that $A_f = \{a_{1,f}, a_{2,f}, b_{0,f}, b_{1,f}\}$ induces a C_4 and $|X \cap A_f| = 2$ since A_f contains two edges of M, hence $X \cap A_f \in \{\{a_{1,f}, b_{1,f}\}, \{a_{2,f}, b_{0,f}\}\}$ for all $f \in [4]$.

We claim that $\mathbf{state}_{(X\cap P)\setminus\{u_1,u_2\}}(a_{1,f}) = \mathbf{s}^\ell_4$ for all $f \in [4]$. Observe that $\mathbf{s}^\ell_2 = \mathbf{0} \Leftrightarrow \mathbf{s}^\ell_4 \neq \mathbf{0}$ and $\mathbf{s}^\ell_1 = \mathbf{s}^\ell_2 \Leftrightarrow \mathbf{s}^\ell_4 \neq \mathbf{1}_0$. Hence, by construction W_ℓ is adjacent to $b_{[\mathbf{s}^\ell_4 \neq \mathbf{0}],f}$ and $c_{[\mathbf{s}^\ell_4 \neq \mathbf{1}_0],f}$, so $b_{[\mathbf{s}^\ell_4 \neq \mathbf{0}],f}, c_{[\mathbf{s}^\ell_4 \neq \mathbf{1}_0],f} \in X$ to cover the edges incident to W_ℓ. So, we see that $a_{1,f} \in X \Leftrightarrow b_{1,f} \in X \Leftrightarrow \mathbf{s}^\ell_4 \neq \mathbf{0}$ as desired. Concerning the root-connectivity of $a_{1,f}$ in $(X \cap P) \setminus \{u_1, u_2\}$, we know that the adjacent vertices $a_{2,f}$ and $b_{0,f}$ are not in X when $a_{1,f}$ is in X, due to A_f inducing a C_4, hence $a_{1,f}$ can only be root-connected via $c_{1,f}$. Since $c_{1,f} \in X \Leftrightarrow \mathbf{s}^\ell_4 \neq \mathbf{1}_0$, the claim follows.

The claim implies that $\mathbf{state}_{X\cap P}(a_{1,4}) = \mathbf{s}^\ell_4$ as desired. We proceed by computing $\mathbf{state}_{(X\cap P)\setminus\{u_1,u_2\}}(\bar{a}_{1,i})$ for $i \in 1, 2$. Due to the degree-1-neighbor $\bar{a}_{2,i}$, we see that $\bar{a}_{1,i} \in X$ because X is a connected vertex cover. The vertex $\bar{a}_{1,i}$ can only be root-connected via $\bar{c}_{1,i}$ and because $\bar{c}_{1,i}$ is an endpoint of a matching edge, we see that $\bar{c}_{1,i} \in X$ if and only if $\bar{c}_{1,i}$ is adjacent to W_ℓ. For $i = 1$, we have that

$$\mathbf{state}_{(X\cap P)\setminus\{u_1,u_2\}}(\bar{a}_{1,1}) = \mathbf{1}_1 \Leftrightarrow \bar{c}_{1,1} \in X \Leftrightarrow \mathbf{s}^\ell_1 \neq \mathbf{1}_0 \Leftrightarrow \ell \in \{1, 3, 5\}.$$

For $i = 2$, we have that

$$\mathbf{state}_{(X\cap P)\setminus\{u_1,u_2\}}(\bar{a}_{1,2}) = \mathbf{1}_1 \Leftrightarrow \bar{c}_{1,2} \in X \Leftrightarrow \mathbf{s}^\ell_1 \neq \mathbf{1}_1 \Leftrightarrow \mathbf{s}^\ell_3 = \mathbf{1}_1.$$

In particular, we have shown that $\mathbf{state}_{X\cap P}(\bar{a}_{1,2}) = \mathbf{s}^\ell_3$ as desired.

It remains to show that $\mathbf{state}_{X\cap P}(u_1) = \mathbf{s}^\ell_1$ and $\mathbf{state}_{X\cap P}(u_2) = \mathbf{s}^\ell_2$. Due to $|X \cap \{u_1, u_2, w_{+,1}, \ldots, w_{+,4}\}| = 2$ and X being canonical, we see that

$$X \cap \{u_1, u_2\} = \begin{cases} \emptyset, & \ell = 1, \\ \{u_1\}, & \ell \in \{2, 3\}, \\ \{u_1, u_2\}, & \ell \in \{4, 5\}. \end{cases}$$

Hence, we only have to determine the root-connectivity of u_1 and possibly u_2 in $X \cap P$ for $\ell > 1$. They can only obtain root-connectivity via $a_{1,1}$, $a_{1,2}$, $a_{1,3}$, or $\bar{a}_{1,1}$. By the previous calculations, at least one of these is root-connected in $(X \cap P) \setminus \{u_1, u_2\}$ if and only if $\mathbf{s}_3^\ell = \mathbf{1}_0$ or $\mathbf{s}_4^\ell = \mathbf{1}_1$, which happens precisely when $\ell \in \{3, 5\}$ as desired (as $\ell = 1$ is excluded). □

To prove that every satisfying assignment of the SATISFIABILITY instance also yields a small connected vertex cover of the constructed graph, we prove that for every state a partial solution attaining this state actually exists. Moreover, it is sufficient to check root-connectivity at the input and output vertices for these partial solutions.

Lemma 8.2.3. *For every $\ell \in [5]$, there exists a canonical vertex cover X_P^ℓ of P such that $|X_P^\ell| = 35$, $X_P^\ell \cap \{v_1, \ldots, v_5\} = \{v_1, \ldots, v_5\} \setminus \{v_\ell\}$, and $\mathbf{state}(X_P^\ell) = \mathbf{s}^\ell$. If X is a vertex cover of G with $\hat{r} \in X$, $X \cap P = X_P^\ell$, and $\mathbf{state}_X(\{u_1, u_2, \bar{a}_{1,2}, a_{1,4}\}) \subseteq \{\mathbf{0}, \mathbf{1}_1\}$, then every vertex of X_P^ℓ is root-connected in X.*

Proof. We claim that

$$X_P^\ell = \left(\bigcup_{k \in [5] \setminus \{\ell\}} W_k \cup \{v_k\} \right) \cup \{\bar{a}_{1,1}, \bar{a}_{1,2}\} \cup \{a_{2-[\mathbf{s}_2^\ell = 0], f} : f \in [4]\} \cup U_\ell \cup N(W_\ell),$$

where $U_1 = \emptyset$, $U_2 = U_3 = \{u_1\}$, $U_4 = U_5 = \{u_1, u_2\}$, is the desired vertex cover. Clearly, X_P^ℓ is canonical. By construction of P, we compute that

$$N(W_\ell) = \{\bar{v}_\ell, \bar{c}_{[s_1^\ell \neq \mathbf{1}_0], 1}, \bar{c}_{[s_1^\ell \neq \mathbf{1}_1], 2}\} \cup \{b_{[\mathbf{s}_2^\ell = 0], f}, c_{[\mathbf{s}_1^\ell = \mathbf{s}_2^\ell], f} : f \in [4]\} \cup W_{+,\ell},$$

where $W_{+,1} = \{w_{+,1}, w_{+,2}\}, W_{+,2} = \{w_{+,3}\}, W_{+,3} = \{w_{+,4}\}, W_{+,4} = W_{+,5} = \emptyset$. Note that $|U_\ell| + |W_{+,\ell}| = 2$ and hence $|X_P^\ell| = 35$ for all $\ell \in [5]$.

We proceed by verifying that X_P^ℓ is a vertex cover of P. The only non-trivial edges to consider are $\{a_{1,f}, c_{1,f}\}$, $f \in [4]$, and the edges between $\{u_1, u_2\}$ and $\{a_{1,f} : f \in [3]\}$. If $a_{1,f} \notin X_P^\ell$, then $\mathbf{s}_2^\ell \neq \mathbf{0}$ which also implies that $\mathbf{s}_1^\ell = \mathbf{s}_2^\ell$ and hence $c_{1,f} \in X_P^\ell$, so the edge $\{a_{1,f}, c_{1,f}\}$, $f \in [4]$, is covered in all cases. If $1 \leq \ell \leq 3$, then $\mathbf{s}_2^\ell = \mathbf{0}$, so $a_{1,f} \in X_P^\ell$ for all $f \in [4]$. If $4 \leq \ell \leq 5$, then $u_1, u_2 \in X$, so in either case the edges between $\{u_1, u_2\}$ and $\{a_{1,f} : f \in [3]\}$ are covered.

Moving on to the second part, assume that X is a vertex cover of G with $\hat{r} \in X$, $X \cap P = X_P^\ell$, and $\mathbf{state}_X(\{u_1, u_2, \bar{a}_{1,2}, a_{1,4}\}) \subseteq \{\mathbf{0}, \mathbf{1}_1\}$. We only have to consider the vertices in $X_P^\ell \setminus N(\hat{r}) \subseteq \{a_{1,f} : f \in [4]\} \cup \{\bar{a}_{1,1}, \bar{a}_{1,2}\}$. The statement immediately follows if u_1 or u_2 is root-connected in X, because they are adjacent to all vertices in $\{a_{1,f} : f \in [3]\} \cup \{\bar{a}_{1,1}\}$ and $a_{1,4}$ and $\bar{a}_{1,2}$ are handled by assumption. It remains to consider the case $u_1, u_2 \notin X$ which corresponds to $\ell = 1$, so we see that $a_{1,f}, c_{1,f} \in X$ for all $f \in [4]$ and $\bar{c}_{1,1} \in X$. Then, $a_{1,f}$ is root-connected via $c_{1,f}$ and $\bar{a}_{1,1}$ is root-connected via $\bar{c}_{1,1}$. □

State Transitions. In the complete construction, we create long paths by repeatedly concatenating the path gadgets P. To study the *state transitions* between two consecutive path gadgets, suppose that we have two copies P^1 and P^2 of P such that the vertices $a_{1,4}$ and $\bar{a}_{1,2}$ in P^1 are joined to the vertices u_1 and u_2 in P^2. We denote the vertices of P^1 with a superscript 1 and the vertices of P^2 with a superscript 2, e.g., $a_{1,4}^1$ refers to the vertex $a_{1,4}$ of

P^1. Again, suppose that P^1 and P^2 are embedded as induced subgraphs in a larger graph G with a root vertex $\hat{r}$ and that only the vertices $u_1, u_2^1, a_{1,4}^2, \bar{a}_{1,2}^2$ and the clique vertices v_ℓ^1, v_ℓ^2, $\ell \in [5]$, have neighbors outside of $P^1 + P^2 + \hat{r}$. Let X be a connected vertex cover of G with $\hat{r} \in X$.

By making use of Lemma 8.2.2 and analyzing the behavior at the boundary between P^1 and P^2, which essentially boils down to computing the entries of the triangular compatibility matrix, we show that the state from one path gadget to the next can only transition according to the desired transition order.

Lemma 8.2.4. *Suppose that X is canonical with respect to $\{u_1^1, u_2^1\}$ and $\{u_2^1, u_2^2\}$, that $G[X]$ is connected and that $|X \cap P^1| \leq 35$ and $|X \cap P^2| \leq 35$, then $\mathbf{state}(X \cap P^1) = \mathbf{s}^{\ell_1}$ and $\mathbf{state}(X \cap P^2) = \mathbf{s}^{\ell_2}$ with $\ell_1 \leq \ell_2$.*

Additionally, for each $\ell \in [5]$, the set $X^\ell = X^\ell_{P^1} \cup X^\ell_{P^2}$ is a vertex cover of $P^1 + P^2$ with $\mathbf{state}_{X^\ell}(\{u_1^1, u_2^1, a_{1,4}^2, \bar{a}_{1,2}^2\}) \subseteq \{\mathbf{0}, \mathbf{1}_1\}$.

Proof. By Lemma 8.2.2, we see that there are $\ell_1, \ell_2 \in [5]$ such that $\mathbf{state}(X \cap P^1) = \mathbf{s}^{\ell_1}$ and $\mathbf{state}(X \cap P^2) = \mathbf{s}^{\ell_2}$. It remains to show that $\ell_1 \leq \ell_2$.

Define $U^1 = \{a_{1,4}^1, \bar{a}_{1,2}^1\}$ and $U^2 = \{u_1^2, u_2^2\}$ and $U = U^1 \cup U^2$. By the assumption on how $P^1 + P^2 + \hat{r}$ can be connected to the rest of the graph G, one can see that any path from U to $\hat{r}$ passes through some vertex in $(V(P_1) \cup V(P_2)) \cap N(\hat{r})$. Hence, we can determine whether the vertices of $X \cap U$ are root-connected in X by just considering the graph $P^1 + P^2 + \hat{r}$.

Consider the state pairs $\bar{\mathbf{s}}^1 = (\mathbf{state}_{X \cap P^1}(\bar{a}_{1,2}^1), \mathbf{state}_{X \cap P^1}(a_{1,4}^2)) = (\mathbf{s}_3^{\ell_1}, \mathbf{s}_4^{\ell_2})$ and $\bar{\mathbf{s}}^2 = (\mathbf{state}_{X \cap P^2}(u_1^2), \mathbf{state}_{X \cap P^2}(u_2^2)) = (\mathbf{s}_1^{\ell_2}, \mathbf{s}_2^{\ell_2})$. We claim that whenever $\ell_1 > \ell_2$ there is some edge in $G[U]$ that is not covered by X or there is a vertex in $X \cap U$ that is not root-connected in X. There is an uncovered edge in $G[U]$ if and only if both $\bar{\mathbf{s}}^1$ and $\bar{\mathbf{s}}^2$ each contain at least one $\mathbf{0}$. This shows that $(\ell_1, \ell_2) \notin \{4, 5\} \times [3]$. Some vertex in $X \cap U$ is not root-connected in X if and only if either $\bar{\mathbf{s}}^1$ or $\bar{\mathbf{s}}^2$ contains a $\mathbf{1}_0$ and the other one only contains two $\mathbf{0}$s or if both contain no $\mathbf{1}_1$ at all. This shows that $(\ell_1, \ell_2) \notin \{(5, 4), (3, 2), (3, 1), (2, 1)\}$ and concludes the proof of the first part.

For the second part, notice that $\mathbf{state}(X^\ell_{P^1}) = \mathbf{state}(X^\ell_{P^2}) = \mathbf{s}^\ell$ by Lemma 8.2.2 and using the same approach as in the last paragraph, we see that for $\ell = \ell_1 = \ell_2$ all edges in $G[U]$ are covered and all vertices in X^ℓ are root-connected in X^ℓ. □

We say that a *cheat occurs* if $\ell_1 < \ell_2$. By Lemma 8.2.4, even when we concatenate arbitrarily many path gadgets P, thus creating a long path, at most $|\mathbf{States}| - 1 = 4 = \mathcal{O}(1)$ cheats may occur on such a path. When proving that a small connected vertex cover gives rise to a satisfying assignment, this allows us to find a *cheat-free region* as outlined before, where we can read off a satisfying assignment.

8.2.2 Complete Construction

Setup. Assume that Connected Vertex Cover can be solved in time $\mathcal{O}^*((5-\varepsilon)^{\text{tc-pw}(G)})$ for some $\varepsilon > 0$. Given a Satisfiability-instance σ with n variables and m clauses, we construct an equivalent Connected Vertex Cover instance with twinclass-pathwidth approximately $n \log_5(2)$ so that the existence of such an algorithm for Connected Vertex Cover would imply that CNF-SETH is false.

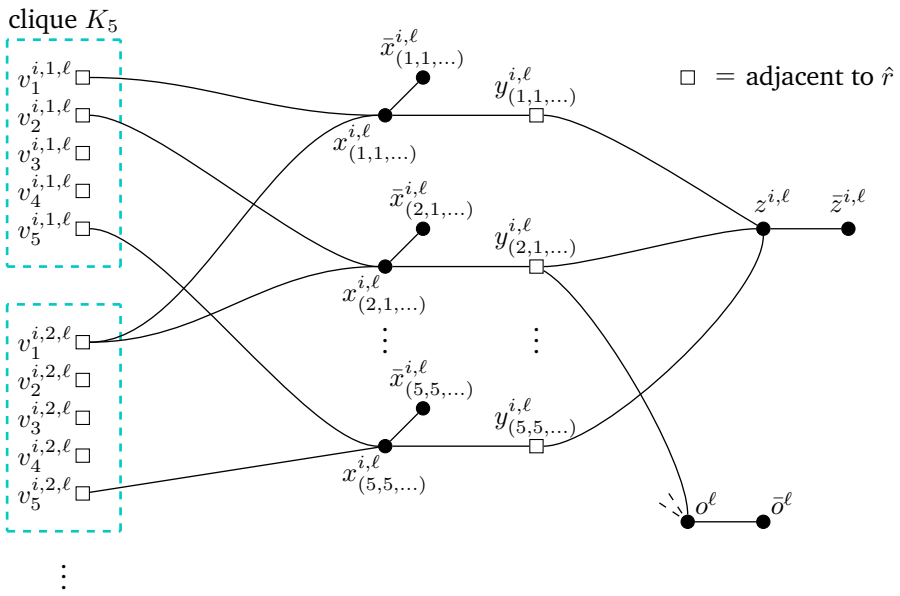


Fig. 8.3.: The decoding gadget for group $i \in [t]$ and column $\ell \in [m(4tp+1)]$. The clause gadget for column ℓ consists of o^ℓ and $\bar{o}^\ell$ and represents clause $C_{\ell'}$, where $\ell' = (\ell - 1) \mod m$. In this figure the truth assignment for group i corresponding to $(2, 1, \ldots) \in [5]^p$ satisfies clause $C_{\ell'}$.

We pick an integer β only depending on ε; the precise choice of β will be discussed at a later point. The variables of σ are partitioned into groups of size at most β, resulting in $t = \lceil n/\beta \rceil$ groups. Furthermore, we pick the smallest integer p that satisfies $5^p \geq 2^\beta$. We now begin with the construction of the CONNECTED VERTEX COVER instance $(G = G(\sigma, \beta), \bar{b})$.

We create the root vertex $\hat{r}$ and attach a leaf $\hat{r}'$ which forces $\hat{r}$ into any connected vertex cover. For every group $i \in [t]$, we create p long path-like gadgets $P^{i,j}$, $j \in [p]$, where each $P^{i,j}$ consists of $m(4tp+1)$ copies $P^{i,j,\ell}$, $\ell \in [m(4tp+1)]$, of the path gadget P and consecutive copies are connected by a join. More precisely, the vertices in some $P^{i,j,\ell}$ inherit their names from P and the superscript of $P^{i,j,\ell}$ and for every $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)-1]$, the output vertices $a_{1,4}^{i,j,\ell}$ and $\bar{a}_{1,2}^{i,j,\ell}$ are joined to the input vertices $u_1^{i,j,\ell+1}$ and $u_2^{i,j,\ell+1}$ of the next path gadget. The ends of each path $P^{i,j}$, i.e., the vertices $u_1^{i,j,1}$, $u_2^{i,j,1}$, $a_{1,4}^{i,j,m(4tp+1)}$, $\bar{a}_{1,2}^{i,j,m(4tp+1)}$ are made adjacent to $\hat{r}$.

For every group $i \in [t]$ and column $\ell \in [m(4tp+1)]$, we create a *decoding gadget* $D^{i,\ell}$ in the same style as Cygan et al. [50] for CONNECTED VERTEX COVER[pathwidth]. Every variable group i has at most 2^β possible truth assignments and by choice of p we have that $5^p \geq 2^\beta$, so we can find an injective mapping $\kappa\colon \{0,1\}^\beta \to [5]^p$ which assigns to each truth assignment $\tau \in \{0,1\}^\beta$ a sequence $\kappa(\tau) \in [5]^p$. For each sequence $\mathbf{h} = (h_1, \ldots, h_p) \in [5]^p$, we create vertices $x_{\mathbf{h}}^{i,\ell}$, $\bar{x}_{\mathbf{h}}^{i,\ell}$, $y_{\mathbf{h}}^{i,\ell}$ and edges $\{x_{\mathbf{h}}^{i,\ell}, \bar{x}_{\mathbf{h}}^{i,\ell}\}$, $\{x_{\mathbf{h}}^{i,\ell}, y_{\mathbf{h}}^{i,\ell}\}$, $\{y_{\mathbf{h}}^{i,\ell}, \hat{r}\}$. Furthermore, we add the edge $\{x_{\mathbf{h}}^{i,\ell}, v_{h_j}^{i,j,\ell}\}$ for all $\mathbf{h} = (h_1, \ldots, h_p) \in [5]^p$ and $j \in [p]$. Finally, we create two adjacent vertices $z^{i,\ell}$ and $\bar{z}^{i,\ell}$ and edges $\{z^{i,\ell}, y_{\mathbf{h}}^{i,\ell}\}$ for all $\mathbf{h} \in [5]^p$. For every group $i \in [t]$ and column $\ell \in [m(4tp+1)]$, we bundle the the path gadgets $P^{i,j,\ell}$, $j \in [p]$, and the decoding gadget $D^{i,\ell}$ into the *block* $B^{i,\ell}$.

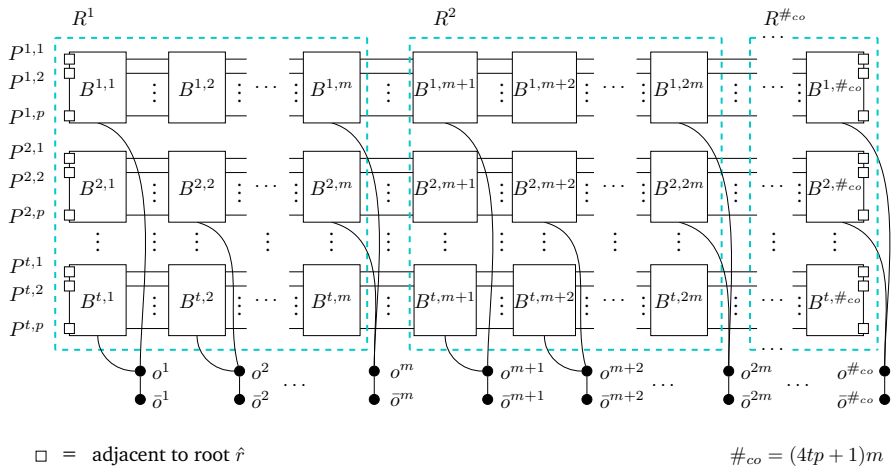


Fig. 8.4.: The matrix structure of the constructed graph for Connected Vertex Cover[twinclass-pathwidth]. Every m columns form a region, which are highlighted by the dashed cyan rectangles.

Lastly, we construct the *clause gadgets*. We number the clauses of σ by $C_0, \ldots, C_{m-1}$. For every column $\ell \in [m(4tp+1)]$, we create an adjacent pair of vertices o^ℓ and $\bar{o}^\ell$. Let $\ell' \in [0, m-1]$ be the remainder of $(\ell - 1)$ modulo m. For every $i \in [t]$, $\mathbf{h} \in \kappa(\{0,1\}^\beta)$, we add the edge $\{o^\ell, y^{i,\ell}_{\mathbf{h}}\}$ whenever $\kappa^{-1}(\mathbf{h})$ is a truth assignment for variable group i that satisfies clause $C_{\ell'}$. See Figure 8.3 for a depiction of the decoding and clause gadgets and Figure 8.4 for a high-level view of the whole construction.

Lemma 8.2.5. *If σ is satisfiable, then there exists a connected vertex cover X of $G = G(\sigma, \beta)$ of size $|X| \leq (35tp + (5^p + 2)t + 1)m(4tp+1) + 1 = \bar{b}$.*

Proof. Let τ be a satisfying truth assignment of σ and let τ^i denote the restriction of τ to the i-th variable group for every $i \in [t]$ and let $\kappa(\tau^i) = \mathbf{h}^i = (h^i_1, \ldots, h^i_p)$ be the corresponding sequence. The connected vertex cover is given by

$$X = \{\hat{r}\} \cup \bigcup_{\ell \in [m(4tp+1)]} \left(\{o^\ell\} \cup \bigcup_{i \in [t]} \left(\{y^{i,\ell}_{\mathbf{h}^i}, z^{i,\ell}\} \cup \bigcup_{\mathbf{h} \in [5]^p} \{x^{i,\ell}_{\mathbf{h}}\} \cup \bigcup_{j \in [p]} X^{h^i_j}_{P^{i,j,\ell}} \right) \right),$$

where $X^{h^i_j}_{P^{i,j,\ell}}$ refers to the sets given by Lemma 8.2.3.

Clearly, $|X| = \bar{b}$, so it remains to prove that X is a connected vertex cover. By Lemma 8.2.3 and the second part of Lemma 8.2.4 all edges induced by the path gadgets are covered by X and all vertices on the path gadgets that belong to X are root-connected, except for possibly the vertices at the ends, i.e. $\bigcup_{i \in [t]} \bigcup_{j \in [p]} \{u^{i,j,1}_1, u^{i,j,1}_2, a^{i,j,m(4tp+1)}_{1,4}, \bar{a}^{i,j,m(4tp+1)}_{1,2}\}$, but these are contained in the neighborhood of $\hat{r}$ by construction.

Fix $i \in [t]$, $\ell \in [m(4tp+1)]$, and consider the corresponding decoding gadget. Since $z^{i,\ell} \in X$ and $x^{i,\ell}_{\mathbf{h}} \in X$ for all $\mathbf{h} \in [5]^p$, all edges induced by the decoding gadget and all edges between the decoding gadget and the path gadgets are covered by X. Furthermore, since $o^\ell \in X$, all edges inside the clause gadget and all edges between the clause gadget and the decoding gadgets are covered by X. Hence, X has to be a vertex cover of G.

It remains to prove that the vertices in the decoding and clause gadgets that belong to X are also root-connected. Again, fix $i \in [t]$, $\ell \in [m(4tp+1)]$, and $\mathbf{h} = (h_1, \ldots, h_p) \in [5]^p \setminus \{\mathbf{h}^i\}$. Since $\mathbf{h} \neq \mathbf{h}^i$, there is some $j \in [p]$ such that $v_{h_j}^{i,j,\ell} \in X$ by Lemma 8.2.3 which connects $x_{\mathbf{h}}^{i,\ell}$ to the root $\hat{r}$. The vertices $x_{\mathbf{h}^i}^{i,\ell}$ and $z^{i,\ell}$ are root-connected via $y_{\mathbf{h}^i}^{i,\ell} \in X$.

We conclude by showing that o^ℓ is root-connected for all $\ell \in [m(4tp+1)]$. Since τ is a satisfying truth assignment of σ, there is some variable group $i \in [t]$ such that τ^i already satisfies clause $C_{\ell'}$, where ℓ' is the remainder of $\ell - 1$ modulo m. By construction of G and X, the vertex $y_{\mathbf{h}^i}^{i,\ell} \in X$ is adjacent to o^ℓ, since $\kappa(\tau^i) = \mathbf{h}^i$, and connects o^ℓ to the root $\hat{r}$. This shows that all vertices of X are root-connected, so $G[X]$ has to be connected. □

Lemma 8.2.6. *If there exists a connected vertex cover X of $G = G(\sigma, \beta)$ of size $|X| \leq (35tp + (5^p + 2)t + 1)m(4tp+1) + 1 = \bar{b}$, then σ is satisfiable.*

Proof. We assume without loss of generality that X is canonical with respect to each twinclass $\{u_1^{i,j,\ell}, u_2^{i,j,\ell}\}$, $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$.

We begin by arguing that X has to satisfy $|X| = \bar{b}$. First, we must have that $\hat{r} \in X$, because $\hat{r}$ has a neighbor of degree 1. By Lemma 8.2.1, we have that $|X \cap P^{i,j,\ell}| \geq 35$ for all $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$. In every decoding gadget, i.e. one for every $i \in [t]$ and $\ell \in [m(4tp+1)]$, the set $\{z^{i,\ell}\} \cup \bigcup_{\mathbf{h} \in [5]^p} x_{\mathbf{h}}^{i,\ell}$ has to be contained in X, since every vertex in this set has a neighbor of degree 1. Furthermore, to connect $z^{i,j}$ to $\hat{r}$, at least one of the vertices $y_{\mathbf{h}}^{i,\ell}$, $\mathbf{h} \in [5]^p$, has to be contained in X. Hence, X must contain at least $5^p + 2$ vertices per decoding gadget. Lastly, $o^\ell \in X$ for all $\ell \in [m(4tp+1)]$, since o^ℓ has a neighbor of degree 1. Since we have only considered disjoint vertex sets, this shows that $|X| = \bar{b}$ and all of the previous inequalities have to be tight, in particular for every $i \in [t]$ and $\ell \in [m(4tp+1)]$, there is a unique $\mathbf{h} \in [5]^p$ such that $y_{\mathbf{h}}^{i,\ell} \in X$.

By Lemma 8.2.2, we know that X assumes one of the five possible states on each $P^{i,j,\ell}$. Fix some $P^{i,j} = \bigcup_{\ell \in [m(4tp+1)]} P^{i,j,\ell}$ and note that due to Lemma 8.2.4 the state can change at most four times along $P^{i,j}$. Such a state change is called a *cheat*. Let $\gamma \in [0, 4tp]$ and define the γ-th *region* $R^\gamma = \bigcup_{i \in [t]} \bigcup_{j \in [p]} \bigcup_{\ell=\gamma m+1}^{(\gamma+1)m} P^{i,j,\ell}$. Since there are $4tp+1$ regions and tp many paths, there is at least one region R^γ such that no cheat occurs in R^γ. We consider region R^γ for the rest of the proof and read off a satisfying truth assignment from this region.

For $i \in [t]$, let $\mathbf{h}^i = (h_1^i, \ldots, h_p^i) \in [5]^p$ such that $v_{h_j^i}^{i,j,\gamma m+1} \notin X$ for all $j \in [p]$; this is well-defined by Lemma 8.2.2. Since R^γ does not contain any cheats, the definition of $\mathbf{h}^i$ is independent of which column $\ell \in [\gamma m + 1, (\gamma+1)m]$ we consider. For every $i \in [t]$ and $\ell \in [\gamma m + 1, (\gamma+1)m]$, we claim that $y_{\mathbf{h}}^{i,\ell} \in X$ if and only if $\mathbf{h} = \mathbf{h}^i$. We have already established that for every i and ℓ, there is exactly one $\mathbf{h}$ such that $y_{\mathbf{h}}^{i,\ell} \in X$. Consider the vertex $x_{\mathbf{h}^i}^{i,\ell} \in X$, its neighbors in G are $v_{h_1^i}^{i,1,\ell}, v_{h_2^i}^{i,2,\ell}, \ldots, v_{h_p^i}^{i,p,\ell}$, $\bar{x}_{\mathbf{h}^i}^{i,\ell}$, and $y_{\mathbf{h}^i}^{i,\ell}$. By construction of $\mathbf{h}^i$ and the tight allocation of the budget, we have $(N(x_{\mathbf{h}^i}^{i,\ell}) \setminus \{y_{\mathbf{h}^i}^{i,\ell}\}) \cap X = \emptyset$. Therefore, X has to include $y_{\mathbf{h}^i}^{i,\ell}$ to connect $x_{\mathbf{h}^i}^{i,\ell}$ to the root $\hat{r}$. This shows the claim.

For $i \in [t]$, we define the truth assignment τ^i for group i by taking an arbitrary truth assignment if $\mathbf{h}^i \notin \kappa(\{0,1\}^\beta)$ and setting $\tau^i = \kappa^{-1}(\mathbf{h}^i)$ otherwise. By setting $\tau = \bigcup_{i \in [t]} \tau^i$ we obtain a truth assignment for all variables and we claim that τ satisfies σ. Consider some clause $C_{\ell'}$, $\ell' \in [0, m-1]$, and let $\ell = \gamma m + \ell' + 1$. We have already argued that $o^\ell \in X$ and to connect o^ℓ to the root $\hat{r}$, there has to be some $y_{\mathbf{h}}^{i,\ell} \in N(o^\ell) \cap X$. By the previous claim,

$\mathbf{h} = \mathbf{h}^i$ for some $i \in [t]$ and therefore τ^i, and also τ, satisfy clause $C_{\ell'}$ due to the construction of G. Because the choice of $C_{\ell'}$ was arbitrary, τ has to be a satisfying assignment of σ. □

Lemma 8.2.7. *The constructed graph $G = G(\sigma, \beta)$ has $\text{tc-pw}(G) \leq tp + 3 \cdot 5^p + \mathcal{O}(1)$ and a path decomposition of $G^q = G/\Pi_{tc}(G)$ of this width can be constructed in polynomial time.*

Proof. By construction, all sets $\{u_1^{i,j,\ell}, u_2^{i,j,\ell}\}$, $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$, are twinclasses. Let G' be the graph obtained by contracting each of these twinclasses, denoting the resulting vertex by $u^{i,j,\ell}$, then G^q is a subgraph of G'. We will show that $\text{tc-pw}(G) = \text{pw}(G^q) \leq \text{pw}(G') \leq tp + 3 \cdot 5^p + \mathcal{O}(1)$ by giving an appropriate strategy for the mixed-search-game on G' and applying Lemma 2.4.31.

Algorithm 5: Mixed-search-strategy for G'

```
1  Place searchers on r̂ and r̂′;
2  Place searchers on u^{i,j,1} for all i ∈ [t], j ∈ [p];
3  for ℓ ∈ [m(4tp + 1)] do
4      Place searchers on o^ℓ and ō^ℓ;
5      for i ∈ [t] do
6          Place searchers on all vertices of the decoding gadget D^{i,ℓ};
7          for j ∈ [p] do
8              Place searchers on all vertices of P^{i,j,ℓ} − {u_1^{i,j,ℓ}, u_2^{i,j,ℓ}};
9              Remove searcher from u^{i,j,ℓ} and place it on u^{i,j,ℓ+1};
10             Remove searchers on P^{i,j,ℓ} − {u_1^{i,j,ℓ}, u_2^{i,j,ℓ}};
11         Remove searchers on D^{i,ℓ};
12     Remove searchers on o^ℓ and ō^ℓ;
```

The mixed-search-strategy for G' described in Algorithm 5 proceeds column by column and group by group in each column. The maximum number of placed searchers occurs on line 8 and is $2 + tp + 2 + (3 \cdot 5^p + 2) + 61$. □

Theorem 8.2.8. *No algorithm can solve* Connected Vertex Cover*, given a path decomposition of $G^q = G/\Pi_{tc}(G)$ of width k, in time $\mathcal{O}^*((5-\varepsilon)^k)$ for some $\varepsilon > 0$, unless CNF-SETH fails.*

Proof. Suppose there is an algorithm $\mathbb{A}$ that solves Connected Vertex Cover in time $\mathcal{O}^*((5-\varepsilon)^k)$ for some $\varepsilon > 0$ given a path decomposition of $G^q = G/\Pi_{tc}(G)$ of width k. Given β, we define $\delta_1 < 1$ such that $(5-\varepsilon)^{\log_5(2)} = 2^{\delta_1}$ and δ_2 such that $(5-\varepsilon)^{1/\beta} = 2^{\delta_2}$. By picking β large enough, we can ensure that $\delta = \delta_1 + \delta_2 < 1$. We show how to solve Satisfiability using $\mathbb{A}$ in time $\mathcal{O}(2^{\delta n}\,\text{poly}(m)) = \mathcal{O}^*(2^{\delta n})$, where n is the number of variables, thus contradicting CNF-SETH, Conjecture 2.1.1.

Given a Satisfiability instance σ, construct $G = G(\sigma, \beta)$ and the path decomposition from Lemma 8.2.7 in polynomial time, as we have $\beta = \mathcal{O}(1)$ and hence $p = \mathcal{O}(1)$. We run

$\mathbb{A}$ on G and return its answer. This is correct by Lemma 8.2.5 and Lemma 8.2.6. Due to Lemma 8.2.7, the running time is

$$\begin{aligned}
&\mathcal{O}^*\left((5-\varepsilon)^{tp+3\cdot 5^p+\mathcal{O}(1)}\right) \leq \mathcal{O}^*\left((5-\varepsilon)^{tp}\right) && \leq \mathcal{O}^*\left((5-\varepsilon)^{\lceil \frac{n}{\beta} \rceil p}\right)\\
&\leq \mathcal{O}^*\left((5-\varepsilon)^{\frac{n}{\beta}p}\right) && \leq \mathcal{O}^*\left((5-\varepsilon)^{\frac{n}{\beta}\lceil \log_5(2^\beta)\rceil}\right) \leq \mathcal{O}^*\left((5-\varepsilon)^{\frac{n}{\beta}\log_5(2^\beta)}(5-\varepsilon)^{\frac{n}{\beta}}\right)\\
&\leq \mathcal{O}^*\left(2^{\delta_1\beta\frac{n}{\beta}}2^{\delta_2 n}\right) && \leq \mathcal{O}^*\left(2^{(\delta_1+\delta_2)n}\right) && \leq \mathcal{O}^*\left(2^{\delta n}\right),
\end{aligned}$$

hence completing the proof. □

8.3 Feedback Vertex Set Lower Bound

This subsection is devoted to proving that FEEDBACK VERTEX SET[twinclass-pathwidth] cannot be solved in time $\mathcal{O}^*((5-\varepsilon)^{\text{tc-pw}(G)})$ for some $\varepsilon > 0$ unless the SETH fails. The main challenge is the design of the path gadget. The decoding gadgets are adapted from the lower bound constructions for ODD CYCLE TRANSVERSAL by Hegerfeld and Kratsch [102], cf. Chapter 13, which rely on *arrows* that are adapted from Lokshtanov et al. [126]. We remark that our construction will rely on false twinclasses and not true twinclasses, because in the algorithm for FEEDBACK VERTEX SET it can already be seen that true twinclasses only admit four distinct states instead of the desired five.

Local Cycles. Nontrivial twinclasses quickly lead to *local cycles*, i.e., whenever a twinclass of size at least two has two or more neighboring vertices, then the twinclass together with its neighbors leads to a cycle. Such local cycles are the main culprit of the additional complexity for FEEDBACK VERTEX SET[twinclass-pathwidth] compared to parameterization by pathwidth or treewidth. To preserve the complexity of the sparse case, we also need to consider more *global cycles* that intersect each twinclass in at most one vertex. To create such global cycles for the path gadgets, we use the following construction.

Root. We create a distinguished vertex $\hat{r}$ called the *root* which will be connected to several vertices throughout the construction. Given a vertex subset $X \subseteq V(G)$ with $\hat{r} \in Y$, we say that a vertex $v \in Y$ is *root-connected* in Y if there is a $v, \hat{r}$-path in $G[Y]$. We will just say *root-connected* if Y is clear from the context. The construction and choice of the budget will ensure that the root vertex $\hat{r}$ cannot be deleted by the desired feedback vertex sets. Such a root has also been used in the FEEDBACK VERTEX SET[pathwidth] lower bound by Cygan et al. [50]. Whenever two root-connected vertices have a path between them that avoids the root, we obtain a cycle. In this way, the root is used to create global cycles spanning multiple path gadgets, which must be destroyed by a feedback vertex set.

Determining a Transition Order. We begin by analyzing the possible *twinclass states* for partial solutions of FEEDBACK VERTEX SET. From the discussion on local cycles, one can observe that it does not matter whether two or more than two vertices remain inside a twinclass after deleting vertices from the graph. Therefore, we restrict our attention to

X_1 vs. X_2	$\mathbf{0}_1$	$\mathbf{0}_0$	$\mathbf{1}_1$	$\mathbf{1}_0$	$\mathbf{2}$
$\mathbf{2}$	1	1	1	1	1
$\mathbf{1}_0$	0	1	1	1	1
$\mathbf{1}_1$	0	1	1	0	1
$\mathbf{0}_0$	0	0	1	1	1
$\mathbf{0}_1$	0	0	0	0	1

Tab. 8.3.: The compatibility matrix for FEEDBACK VERTEX SET with respect to a join between two twinclasses.

X_1 vs. X_2	$\mathbf{0}_1$	$(\mathbf{1}_1, \mathbf{1}_0)$	$\mathbf{0}_0$	$\mathbf{1}_0$	$\mathbf{2}$
$\mathbf{2}$	1	1	1	1	1
$\mathbf{1}_0$	0	1	1	1	1
$\mathbf{1}_1$	0	0	1	1	1
$\mathbf{0}_0$	0	0	0	1	1
$\mathbf{0}_1$	0	0	0	0	1

Tab. 8.4.: The triangular compatibility matrix for FEEDBACK VERTEX SET[twinclass-pathwidth] obtained by using asymmetric states.

twinclasses of size two. We describe the state of a twinclass C with respect to a partial solution $X \subseteq V(G)$ by the number of vertices in the intersection $X \cap C$, when $|X \cap C| = 1$ we distinguish whether the remaining vertex in $C \setminus X$ is root-connected (state $\mathbf{1}_1$) in $Y = V(G) \setminus X$ or not (state $\mathbf{1}_0$), and when $X \cap C = \emptyset$ we distinguish whether the two vertices in $C \setminus X$ are connected (state $\mathbf{0}_1$) in $Y = V(G) \setminus X$ or not (state $\mathbf{0}_0$); the remaining state $\mathbf{2}$ represents $|X \cap C| = 2$. First, we look at the compatibility matrix for the join between two twinclasses. Two twinclass states are incompatible when they lead to a local cycle, e.g., $\mathbf{2}_0$ vs. $\mathbf{2}_0$ or $\mathbf{2}_1$ vs. $\mathbf{1}_0$, or a cycle through the root, i.e., $\mathbf{1}_1$ vs. $\mathbf{1}_1$. We obtain the compatibility matrix in Table 8.3, which has rank 5, but cannot be arranged to a triangular matrix of size 5×5: for the bottom two rows of the triangular matrix, we need two rows containing at most two 1s, but there is only one such row in Table 8.3.

Introducing Asymmetry. Our analysis shows again that using a single twinclass at both boundaries will not be sufficient to prove a tight lower bound for FEEDBACK VERTEX SET[twinclass-pathwidth]. Instead, we again create an asymmetric path gadget that uses several twinclasses at the right boundary. This allows us to consider the *heterogeneous state* $(\mathbf{1}_1, \mathbf{1}_0)$ where in one twinclass a root-connected vertex remains and in another a non-root-connected vertex remains. Using the heterogeneous state $(\mathbf{1}_1, \mathbf{1}_0)$ instead of state $\mathbf{1}_1$ at the right boundary, we obtain the triangular compatibility matrix of size 5×5 in Table 8.4, which guides the design of the path gadget. We remark that in the actual construction of the path gadget, we will just use twinclasses of size one, i.e., single vertices, instead of larger ones at the right boundary.

Complete Construction

Triangle Edges. Given two vertices u and v, by *adding a triangle edge between u and v* we mean that we add a new vertex $w_{\{u,v\}}$ and the edges $\{u, v\}$, $\{u, w_{\{u,v\}}\}$, $\{w_{\{u,v\}}, v\}$, so that the three vertices u, v, $w_{\{u,v\}}$ induce a triangle. The vertex $w_{\{u,v\}}$ will not receive any further neighbors in the construction. Any feedback vertex set X has to intersect $\{u, v, w_{\{u,v\}}\}$ and since $w_{\{u,v\}}$ has only degree 2, we can always assume that $w_{\{u,v\}} \notin X$. In this way, a triangle edge naturally implements a logical or between u and v.

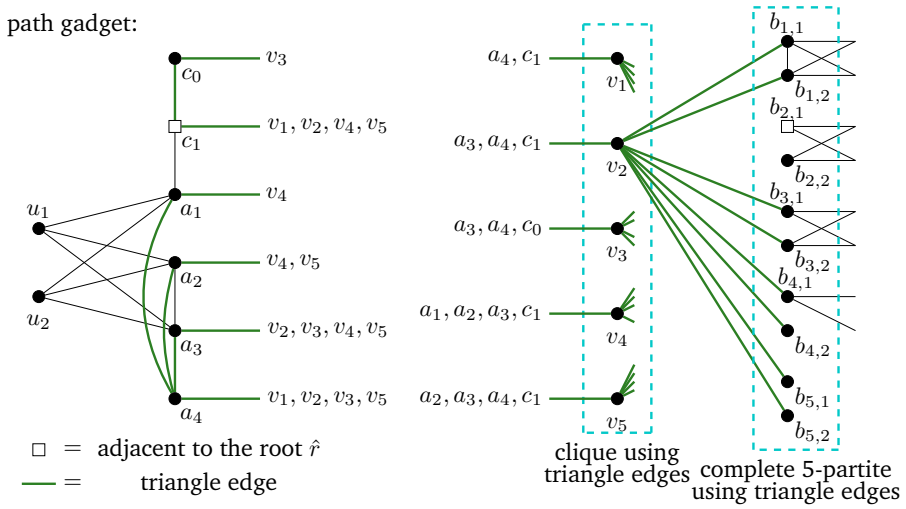


Fig. 8.5.: The superscripts in vertex names are omitted and the edges between the auxiliary vertices, connectivity vertices, and clique vertices are not drawn directly for visual clarity. All vertices that are depicted with a rectangle are adjacent to the root vertex $\hat{r}$. The thick green edges denote triangle edges. The vertices inside the dashed rectangle induce a 5-clique or a complete 5-partite graph using triangle edges. The edges from the output vertices to the next pair of input vertices are hinted at.

Arrows. Given two vertices u and v, by *adding an arrow from u to v* we mean that we add three vertices x_{uv}, y_{uv}, z_{uv} and the edges $\{u, x_{uv}\}$, $\{u, y_{uv}\}$, $\{x_{uv}, y_{uv}\}$, $\{y_{uv}, z_{uv}\}$, $\{y_{uv}, v\}$, $\{z_{uv}, v\}$, i.e., we are essentially adding two consecutive triangle edges between u and v. The resulting graph is denoted by $A(u, v)$ and u is the *tail* and v the *head* of the arrow. None of the vertices in $V(A(u, v)) \setminus \{u, v\}$ will receive any further neighbors in the construction. The construction of an arrow is symmetric, but the direction will be relevant for constructing a cycle packing that witnesses a lower bound on the size of a feedback vertex set.

We use arrows to propagate deletions throughout the graph. Let X be a feedback vertex set. If $u \notin X$, then we can resolve both triangles simultaneously by putting y_{uv} into X. If $u \in X$, then the first triangle is already resolved and we can safely put v into X, hence propagating the deletion from u to v. The former solution is called the *passive* solution of the arrow and the latter is the *active* solution. Using simple exchange arguments, we see that it is sufficient to only consider feedback vertex sets that on each arrow either use the passive solution or the active solution.

Setup of Construction. Assume that Feedback Vertex Set can be solved in time $\mathcal{O}^*((5-\varepsilon)^{\text{tc-pw}(G)})$ for some $\varepsilon > 0$. Given a q-Satisfiability-instance σ with n variables and m clauses, we construct an equivalent Feedback Vertex Set instance with twinclass-pathwidth approximately $n \log_5(2)$ so that the existence of such an algorithm for Feedback Vertex Set would imply that SETH is false. We pick an integer β only depending on ε; the precise choice of β will be discussed at a later point. The variables of σ are partitioned into groups of size at most β, resulting in $t = \lceil n/\beta \rceil$ groups. Furthermore, we pick the smallest integer p that satisfies $5^p \geq 2^\beta$. We now begin with the construction of the FVS instance $(G = G(\sigma, \beta), \overline{b})$; we recall that we first create a root vertex $\hat{r}$.

Path Gadgets. For every $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$, we create a path gadget $P^{i,j,\ell}$ that consists of two *input* vertices $u_1^{i,j,\ell}$, $u_2^{i,j,\ell}$ forming a false twinclass; four *auxiliary* vertices $a_1^{i,j,\ell}, \ldots, a_4^{i,j,\ell}$; two *connectivity* vertices $c_0^{i,j,\ell}$, $c_1^{i,j,\ell}$; five *clique* vertices $v_1^{i,j,\ell}, \ldots, v_5^{i,j,\ell}$; and ten *output* vertices in pairs of two $b_{1,1}^{i,j,\ell}$, $b_{1,2}^{i,j,\ell}$, $b_{2,1}^{i,j,\ell}$, $b_{2,2}^{i,j,\ell}, \ldots, b_{5,2}^{i,j,\ell}$. We add a join between the input vertices $u_1^{i,j,\ell}$, $u_2^{i,j,\ell}$ and the first three auxiliary vertices $a_1^{i,j,\ell}$, $a_2^{i,j,\ell}$, $a_3^{i,j,\ell}$, furthermore we add the edges $\{a_2^{i,j,\ell}, a_3^{i,j,\ell}\}$, $\{a_1^{i,j,\ell}, c_1^{i,j,\ell}\}$, and $\{b_{1,1}^{i,j,\ell}, b_{1,2}^{i,j,\ell}\}$. The vertices $c_1^{i,j,\ell}$ and $b_{2,1}^{i,j,\ell}$ are made adjacent to the root $\hat{r}$. We add triangle edges between $a_4^{i,j,\ell}$ and the other auxiliary vertices $a_1^{i,j,\ell}$, $a_2^{i,j,\ell}$, $a_3^{i,j,\ell}$ and we add a triangle edge between $c_0^{i,j,\ell}$ and $c_1^{i,j,\ell}$. We add a triangle edge between every pair of distinct clique vertices $v_\varphi^{i,j,\ell}$, $\varphi \in [5]$, and every pair of output vertices $b_{\varphi,\gamma}^{i,j,\ell}$ and $b_{\varphi',\gamma'}^{i,j,\ell}$ with $\varphi \neq \varphi' \in [5]$ and $\gamma, \gamma' \in \{1,2\}$. For all $\varphi \in [5]$, we add a triangle edge between $v_\varphi^{i,j,\ell}$ and every $b_{\psi,\gamma}^{i,j,\ell}$ for $\psi \in [5] \setminus \{\varphi\}$ and $\gamma \in \{1,2\}$. We finish the construction of $P^{i,j,\ell}$ by describing how to connect the clique vertices $v_\varphi^{i,j,\ell}$, $\varphi \in [5]$, to the left side of $P^{i,j,\ell}$. For each $\varphi \in [5]$, we add triangle edges between $v_\varphi^{i,j,\ell}$ and one or several *target* vertices on the left side of $P^{i,j,\ell}$. The target vertices, depending on $\varphi \in [5]$, are

- for $\varphi = 1$: $a_4^{i,j,\ell}$ and $c_1^{i,j,\ell}$;
- for $\varphi = 2$: $a_3^{i,j,\ell}$, $a_4^{i,j,\ell}$, and $c_1^{i,j,\ell}$;
- for $\varphi = 3$: $a_3^{i,j,\ell}$, $a_4^{i,j,\ell}$, and $c_0^{i,j,\ell}$;
- for $\varphi = 4$: $a_1^{i,j,\ell}$, $a_2^{i,j,\ell}$, $a_3^{i,j,\ell}$, and $c_1^{i,j,\ell}$;
- for $\varphi = 5$: $a_2^{i,j,\ell}$, $a_3^{i,j,\ell}$, $a_4^{i,j,\ell}$, and $c_1^{i,j,\ell}$.

Finally, for $\ell \in [m(4tp+1)-1]$, we connect $P^{i,j,\ell}$ to $P^{i,j,\ell+1}$ by adding a join between the output pair $b_{\varphi,1}^{i,j,\ell}$, $b_{\varphi,2}^{i,j,\ell}$ and the next input vertices $u_1^{i,j,\ell+1}$, $u_2^{i,j,\ell+1}$ for every $\varphi \in \{1,2,3\}$ and we join the vertex $b_{4,1}^{i,j,\ell}$ to $u_1^{i,j,\ell+1}$ and $u_2^{i,j,\ell+1}$. This concludes the description of the path gadgets, cf. Figure 8.5.

Intuition for Path Gadget. The left boundary is built in such a way that a feedback vertex set has to delete at least three of the vertices among the input and auxiliary vertices, so the auxiliary vertices balance out the cost of the deletions at the separating twinclass. Using triangle edges, we force specific vertices into the feedback vertex set depending on which clique vertex is avoided. The construction at the right boundary ensures that all output vertices except the pair corresponding to the avoided clique vertex are deleted. In this way, we have built a generic *selector gadget* choosing the appropriate state at the right boundary via the pairs instead of building a more intricate gadget that forces the desired state at two fixed vertices. The states at the left boundary associated with the avoided clique vertex are $\mathbf{s}^1 = \mathbf{2}$, $\mathbf{s}^2 = \mathbf{1}_0$, $\mathbf{s}^3 = \mathbf{1}_1$, $\mathbf{s}^4 = \mathbf{0}_0$, and $\mathbf{s}^5 = \mathbf{0}_1$, with the same meaning as in the derivation of the transition order. However, in this gadget the relation between the avoided clique vertex and the state at the left boundary is not perfect in the sense that avoiding the i-th clique vertex does not force state $\mathbf{s}^i$ at the left boundary. Instead, we will prove that avoiding the i-th clique vertex at least ensures that the state at the left boundary must be $\mathbf{s}^j$ with $i \leq j$, which preserves the transition order and still allows the proof to go through.

Decoding Gadgets. For every group $i \in [t]$, column $\ell \in [m(4tp+1)]$, and state sequence $\mathbf{h} = (h_1, \ldots, h_p) \in [5]^p$, we create a decoding gadget $D^{i,\ell,\mathbf{h}}$ consisting of $4p$ vertices $x_1^{i,\ell,\mathbf{h}}, \ldots, x_{4p}^{i,\ell,\mathbf{h}}$; a distinguished vertex $\hat{x}^{i,\ell,\mathbf{h}}$; and two vertices $y_1^{i,\ell,\mathbf{h}}$ and $y_2^{i,\ell,\mathbf{h}}$. We add the edges $\{y_1^{i,\ell,\mathbf{h}}, y_2^{i,\ell,\mathbf{h}}\}$, $\{y_1^{i,\ell,\mathbf{h}}, \hat{x}^{i,\ell,\mathbf{h}}\}$, $\{y_2^{i,\ell,\mathbf{h}}, \hat{x}^{i,\ell,\mathbf{h}}\}$ and for every $\gamma \in [4p]$, the edges $\{y_1^{i,\ell,\mathbf{h}}, x_\gamma^{i,\ell,\mathbf{h}}\}$

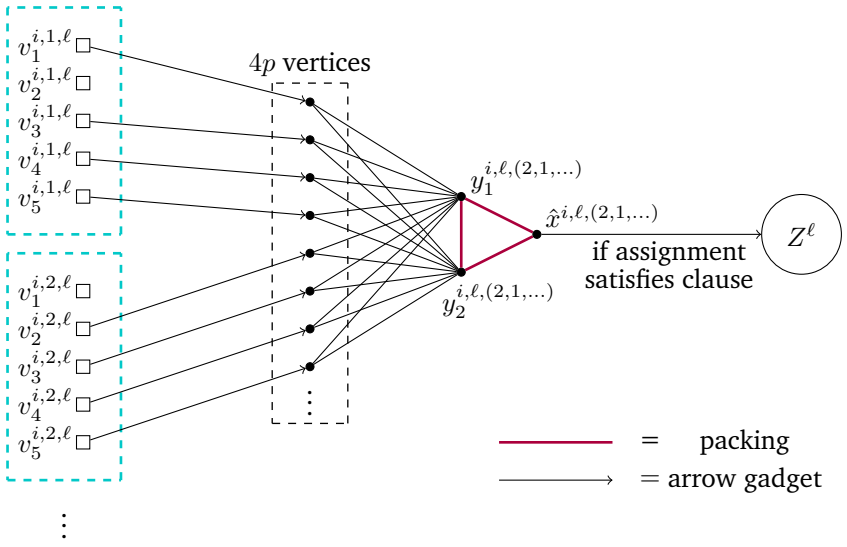


Fig. 8.6.: A depiction of the decoding $D^{i,\ell,\mathbf{h}}$ and clause gadget Z^ℓ with $\mathbf{h} = (2, 1, \ldots)$. The red triangle is part of the packing $\mathcal{P}$. The arrows point in the direction of the deletion propagation.

and $\{y_2^{i,\ell,\mathbf{h}}, x_\gamma^{i,\ell,\mathbf{h}}\}$, hence $\{y_1^{i,\ell,\mathbf{h}}, y_2^{i,\ell,\mathbf{h}}, x_\gamma^{i,\ell,\mathbf{h}}\}$ induces a triangle for every $\gamma \in [4p]$. The path gadgets $P^{i,j,\ell}$ with $j \in [p]$ are connected to $D^{i,\ell,\mathbf{h}}$ as follows. For every clique vertex $v_\varphi^{i,j,\ell}$ with $\varphi \in [5] \setminus \{h_j\}$, we pick a private vertex $x_\gamma^{i,\ell,\mathbf{h}}$, $\gamma \in [4p]$, and add an arrow from $v_\varphi^{i,j,\ell}$ to $x_\gamma^{i,\ell,\mathbf{h}}$. Since there are precisely $4p$ such $v_\varphi^{i,j,\ell}$ for fixed i, ℓ, and $\mathbf{h}$, this construction works out. For every $i \in [t]$, $\ell \in [m(4tp+1)]$, the *block* $B^{i,\ell}$ consists of the path gadgets $P^{i,j,\ell}$, $j \in [p]$, and the decoding gadgets $D^{i,\ell,\mathbf{h}}$, $\mathbf{h} \in [5]^p$. See Figure 8.6 for a depiction of the decoding gadget.

Mapping Truth Assignments to State Sequences. Every variable group $i \in [t]$ has at most 2^β possible truth assignments. By choice of p, we have that $5^p \geq 2^\beta$, hence we can fix an injective mapping $\kappa\colon \{0,1\}^\beta \to [5]^p$ that maps truth assignments $\tau \in \{0,1\}^\beta$ to state sequences $\mathbf{h} \in [5]^p$.

Clause Cycles. We number the clauses of σ by $C_0, \ldots, C_{m-1}$. For every column $\ell \in [m(4tp+1)]$, we create a cycle Z^ℓ consisting of $q5^p$ vertices z_γ^ℓ, $\gamma \in [q5^p]$. Let ℓ' be the remainder of $\ell - 1$ modulo m. For every group $i \in [t]$ and state sequence $\mathbf{h} \in [5]^p$, we add an arrow from $\hat{x}^{i,\ell,\mathbf{h}}$ to a private z_γ^ℓ if $\mathbf{h} \in \kappa(\{0,1\}^\beta)$ and $\kappa^{-1}(\mathbf{h})$ is a truth assignment for variable group i that satisfies clause $C_{\ell'}$. Since σ is a q-SATISFIABILITY instance, every clause intersects at most q variable groups. Every variable group has at most $2^\beta \leq 5^p$ possible truth assignments, hence $q5^p$ is a sufficient number of vertices for this construction to work. See Figure 8.7 for a depiction of the high-level structure.

Packing. We construct a vertex-disjoint packing $\mathcal{P}$ that will witness a lower bound on the size of any feedback vertex set in the constructed graph G. The packing $\mathcal{P}$ consists of the following subgraphs:

- the triangle edge between $c_0^{i,j,\ell}$ and $c_1^{i,j,\ell}$ for all $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$,

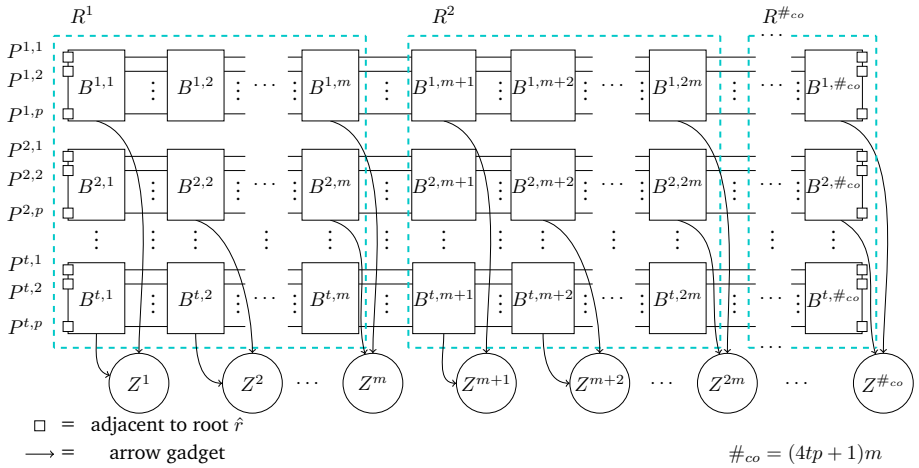


Fig. 8.7.: The matrix structure of the constructed graph for FEEDBACK VERTEX SET[twinclass-pathwidth]. Every m columns form a region, which are highlighted by the dashed cyan rectangles.

- the graph induced by the clique vertices $v_\varphi^{i,j,\ell}$, $\varphi \in [5]$, and the triangle edges between them for all $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$,
- the graph induced by the output vertices $b_{\varphi,\gamma}^{i,j,\ell}$, $\varphi \in [5]$, $\gamma \in \{1,2\}$, and the triangle edges between them for all $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$,
- the graph induced by the input vertices $u_1^{i,j,\ell}$, $u_2^{i,j,\ell}$ and the auxiliary vertices $a_1^{i,j,\ell}$, …, $a_4^{i,j,\ell}$ and the triangle edges between them for all $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$,
- the triangle induced by $\hat{x}^{i,\ell,\mathbf{h}}, y_1^{i,\ell,\mathbf{h}}, y_2^{i,\ell,\mathbf{h}}$ for all $i \in [t]$, $\ell \in [m(4tp+1)]$, $\mathbf{h} \in [5]^p$,
- the second triangle in every arrow $A(u,v)$, i.e., the triangle containing the head v if the arrow was constructed from u to v.

Observe that in the construction of G at most the tail of an arrow is incident with any of the other subgraphs in $\mathcal{P}$, hence the subgraphs in $\mathcal{P}$ are indeed vertex-disjoint. Let n_A be the number of arrows in G, we define

$$\mathrm{cost}_{\mathcal{P}} = (1+4+8+3)tpm(4tp+1) + tm(4tp+1)5^p + n_A.$$

Lemma 8.3.1. *Let X be a feedback vertex set of G, then $|X| \geq \mathrm{cost}_{\mathcal{P}}$.*

Proof. We first apply the standard exchange arguments for triangle edges and arrows to X, obtaining a feedback vertex set X' of G with $|X'| \leq |X|$ that never contains the degree-2 vertex in a triangle edge and always uses the passive or active solution on any arrow.

For every triangle in $\mathcal{P}$, the feedback vertex set X' must clearly contain at least one vertex of that triangle. Fix $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$ for the rest of the proof. Consider the graph induced by the clique vertices $v_\varphi^{i,j,\ell}$, $\varphi \in [5]$, and suppose that there are $\varphi \neq \psi \in [5]$ such that $v_\varphi^{i,j,\ell}, v_\psi^{i,j,\ell} \notin X'$, then the triangle edge between these two vertices is not resolved by assumption on X'. Hence, X' contains at least four of the vertices $v_\varphi^{i,j,\ell}$, $\varphi \in [5]$. Similarly, consider the graph induced by the output vertices $b_{\varphi,\gamma}^{i,j,\ell}$, $\varphi \in [5]$, $\gamma \in \{1,2\}$, and suppose that there are $\varphi \neq \psi \in [5]$, $\gamma, \gamma' \in \{1,2\}$ such that $b_{\varphi,\gamma}^{i,j,\ell}, b_{\psi,\gamma'}^{i,j,\ell} \notin X'$, then the triangle edge between these two vertices is not resolved by assumption on X'. Hence, X' contains at least eight of these vertices, in particular four out of five pairs $b_{\varphi,1}^{i,j,\ell}, b_{\varphi,2}^{i,j,\ell}$, $\varphi \in [5]$, must be completely contained in X'.

It remains to show that X' contains at least three vertices in the subgraph induced by the input vertices $u_1^{i,j,\ell}$, $u_2^{i,j,\ell}$ and the auxiliary vertices $a_1^{i,j,\ell}$, …, $a_4^{i,j,\ell}$. First, observe that X' has to contain all of the first three auxiliary vertices $a_1^{i,j,\ell}$, $a_2^{i,j,\ell}$, $a_3^{i,j,\ell}$ or the last auxiliary vertex $a_4^{i,j,\ell}$, otherwise there is an unresolved triangle edge incident to the last auxiliary vertex $a_4^{i,j,\ell}$. We distinguish three cases based on $\alpha = |X' \cap \{u_1^{i,j,\ell}, u_2^{i,j,\ell}\}|$. If $\alpha = 2$, we are done by the first observation. If $\alpha = 1$, there is a triangle induced by $a_2^{i,j,\ell}$, $a_3^{i,j,\ell}$, and the remaining input vertex which needs to be resolved. Hence, $a_2^{i,j,\ell} \in X'$ or $a_3^{i,j,\ell} \in X'$ and due to the first observation X' has to contain at least one further vertex. Finally, if $\alpha = 0$, note that the graph induced by the input vertices and the first three auxiliary vertices contains a $K_{2,3}$, so X' has to contain at least two of the first three auxiliary vertices and due to the first observation X' has to contain at least one further vertex, hence we are done. □

Lemma 8.3.2. *If σ is satisfiable, then there is a feedback vertex set X of G with $|X| \leq \mathrm{cost}_{\mathcal{P}}$.*

Proof. Let τ be a satisfying truth assignment of σ and let τ^i be the induced truth assignment for variable group $i \in [t]$. Each truth assignment τ^i corresponds to a state sequence $\kappa(\tau^i) = \mathbf{h}^i = (h_1^i, \ldots, h_p^i)$ which we will use to construct the feedback vertex set X. On every path gadget $P^{i,j,\ell}$, $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$, we consider five different types of solutions $X_\varphi^{i,j,\ell}$, $\varphi \in [5]$, which we will define now:

- $X_1^{i,j,\ell} = \{v_\varphi^{i,j,\ell}, b_{\varphi,1}^{i,j,\ell}, b_{\varphi,2}^{i,j,\ell} : \varphi \in [5] \setminus \{1\}\} \cup \{c_1^{i,j,\ell}\} \cup \{a_4^{i,j,\ell}\} \cup \{u_1^{i,j,\ell}, u_2^{i,j,\ell}\}$
- $X_2^{i,j,\ell} = \{v_\varphi^{i,j,\ell}, b_{\varphi,1}^{i,j,\ell}, b_{\varphi,2}^{i,j,\ell} : \varphi \in [5] \setminus \{2\}\} \cup \{c_1^{i,j,\ell}\} \cup \{a_3^{i,j,\ell}, a_4^{i,j,\ell}\} \cup \{u_1^{i,j,\ell}\}$
- $X_3^{i,j,\ell} = \{v_\varphi^{i,j,\ell}, b_{\varphi,1}^{i,j,\ell}, b_{\varphi,2}^{i,j,\ell} : \varphi \in [5] \setminus \{3\}\} \cup \{c_0^{i,j,\ell}\} \cup \{a_3^{i,j,\ell}, a_4^{i,j,\ell}\} \cup \{u_1^{i,j,\ell}\}$
- $X_4^{i,j,\ell} = \{v_\varphi^{i,j,\ell}, b_{\varphi,1}^{i,j,\ell}, b_{\varphi,2}^{i,j,\ell} : \varphi \in [5] \setminus \{4\}\} \cup \{c_1^{i,j,\ell}\} \cup \{a_1^{i,j,\ell}, a_2^{i,j,\ell}, a_3^{i,j,\ell}\} \cup \emptyset$
- $X_5^{i,j,\ell} = \{v_\varphi^{i,j,\ell}, b_{\varphi,1}^{i,j,\ell}, b_{\varphi,2}^{i,j,\ell} : \varphi \in [5] \setminus \{5\}\} \cup \{c_1^{i,j,\ell}\} \cup \{a_2^{i,j,\ell}, a_3^{i,j,\ell}, a_4^{i,j,\ell}\} \cup \emptyset$

The feedback vertex set on the path gadgets $P^{i,j,\ell}$ is given by

$$X_P = \bigcup_{i \in [t]} \bigcup_{j \in [p]} \bigcup_{\ell \in [m(4tp+1)]} X_{h_j^i}^{i,j,\ell}.$$

On the decoding gadgets $D^{i,\ell,\mathbf{h}}$, we define

$$X_D = \bigcup_{i \in [t]} \bigcup_{\ell \in [m(4tp+1)]} \left(\{\hat{x}^{i,\ell,\mathbf{h}^i}\} \cup \{y_1^{i,\ell,\mathbf{h}} : \mathbf{h} \in [5]^p \setminus \{\mathbf{h}^i\}\} \right).$$

We obtain the desired feedback vertex set X by starting with $X_P \cup X_D$ and propagating the deletions throughout G using the arrows, i.e., if the tail u of an arrow $A(u,v)$ is in X, then we choose the active solution on this arrow and otherwise we choose the passive solution. Since $|X_\varphi^{i,j,\ell}| = 16$ for all $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$, $\varphi \in [5]$, we compute that $|X_P| = 16tpm(4tp+1)$ and for X_D, we see that $|X_D| = tm(4tp+1)5^p$ and hence $|X| = \mathrm{cost}_{\mathcal{P}}$ as desired, since we perform one additional deletion per arrow.

It remains to show that X is a feedback vertex set of G, i.e., that $G - X$ is a forest. First, notice that the passive solution of an arrow $A(u,v)$ disconnects u from v inside $A(u,v)$ and that the remainder of $A(u,v) - \{u,v\}$ cannot partake in any cycles. The active solution of an arrow $A(u,v)$ deletes u and v, so that the three remaining vertices of the arrow form a single connected component. Since the path gadgets $P^{i,j,\ell}$ are connected to the decoding gadgets $D^{i,\ell,\mathbf{h}}$ only via arrows and also the decoding gadgets are only connected to the

clause cycles Z^ℓ via arrows, X disconnects these three types of gadgets from each other and we can handle each type separately.

We begin with the decoding gadgets $D^{i,\ell,\mathbf{h}}$, $i \in [t]$, $\ell \in [m(4tp+1)]$, $\mathbf{h} \in [5]^p$. Every $D^{i,\ell,\mathbf{h}}$ is in its own connected component in $G - X$, since one can only enter or leave $D^{i,\ell,\mathbf{h}}$ via an arrow. Every cycle in $D^{i,\ell,\mathbf{h}}$ intersects $y_1^{i,\ell,\mathbf{h}}$ which is in X if $\mathbf{h} \neq \mathbf{h}^i$. Hence, it remains to consider the case $\mathbf{h} = \mathbf{h}^i$. In this case, X contains $\hat{x}^{i,\ell,\mathbf{h}^i}$ by definition of X_D and we claim that $x_\gamma^{i,\ell,\mathbf{h}^i} \in X$ for all $\gamma \in [4p]$ due to propagation via arrows. By construction of G, every $x_\gamma^{i,\ell,\mathbf{h}^i}$, $\gamma \in [4p]$, is the head of an arrow $A(v_\varphi^{i,j,\ell}, x_\gamma^{i,\ell,\mathbf{h}^i})$ for some $j \in [p]$ and $\varphi \in [5] \setminus \{h_p^i\}$, but every such $v_\varphi^{i,j,\ell}$ is in X by definition of X_P. Hence, these deletions are propagated to the $x_\gamma^{i,\ell,\mathbf{h}^i}$, $\gamma \in [4p]$ and the only remaining vertices of $D^{i,\ell,\mathbf{h}^i}$ are $y_1^{i,\ell,\mathbf{h}^i}$ and $y_2^{i,\ell,\mathbf{h}^i}$ which clearly induce an acyclic graph.

We continue with the clause cycles Z^ℓ, $\ell \in [m(4tp+1)]$. Again, each clause cycle Z^ℓ is in its own connected component in $G - X$ and Z^ℓ consists of a single large cycle with vertices z_γ^ℓ, $\gamma \in [q5^p]$. We claim that X propagates a deletion to at least one of these z_γ^ℓ. Let ℓ' be the remainder of $\ell - 1$ modulo m. Since τ satisfies σ and in particular clause $C_{\ell'}$, there is some variable group $i \in [t]$ such that already τ^i satisfies clause $C_{\ell'}$. By construction of G, there is an arrow $A(\hat{x}^{i,\ell,\mathbf{h}^i}, z_\gamma^\ell)$ for some $\gamma \in [q5^p]$ because $\kappa(\tau^i) = \mathbf{h}^i$. By definition of X_D, we have that $\hat{x}^{i,\ell,\mathbf{h}^i} \in X$ and a deletion is indeed propagated to z_γ^ℓ, thus resolving the cycle Z^ℓ.

It remains to show that there is no cycle in $G - X$ intersecting a path gadget $P^{i,j,\ell}$, $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$. All path gadgets are connected to each other via the root vertex $\hat{r}$ and furthermore consecutive path gadgets $P^{i,j,\ell}$ and $P^{i,j,\ell+1}$ are connected via the joins between them. We first show that there is no cycle in $G - X$ that is completely contained in a single path gadget $P^{i,j,\ell}$. It is easy to see that each $X \cap P^{i,j,\ell} = X_{h_j^i}^{i,j,\ell}$ contains at least one vertex per triangle edge in $P^{i,j,\ell}$. Any further cycle that could remain in $P^{i,j,\ell}$ can only involve the vertices $u_1^{i,j,\ell}$, $u_2^{i,j,\ell}$, $a_1^{i,j,\ell}$, $a_2^{i,j,\ell}$, and $a_3^{i,j,\ell}$. These vertices induce a $K_{2,3}$ plus the edge $\{a_2^{i,j,\ell}, a_3^{i,j,\ell}\}$ in G. In each $X_\varphi^{i,j,\ell}$, $\varphi \in [5]$, one side of the biclique $K_{2,3}$ is contained completely with the exception of at most one vertex and $a_2^{i,j,\ell}$ and $a_3^{i,j,\ell}$ only remain together if the other side is contained completely. Hence, no cycle remains there either.

Observe that $P^{i,j,\ell}$ is separated from any $P^{i,j,\ell'}$ with $\ell' \notin \{\ell-1, \ell, \ell+1\}$ in $G - (X \cup \{\hat{r}\})$, because X contains at least one endpoint of each triangle edge between the clique vertices $v_\varphi^{i,j,\ell}$, $\varphi \in [5]$, and the output vertices $b_{\varphi,\gamma}^{i,j,\ell}$, $\varphi \in [5]$, $\gamma \in \{1,2\}$. Hence, any cycle in $G - (X \cup \{\hat{r}\})$ would have to involve two consecutive path gadgets. Furthermore, $\{u_1^{i,j,\ell+1}, u_2^{i,j,\ell+1}\}$ is a separator of size two between $P^{i,j,\ell}$ and $P^{i,j,\ell+1}$ in $G - (X \cup \{\hat{r}\})$, so any cycle involving both path gadgets has to contain $u_1^{i,j,\ell+1}$ and $u_2^{i,j,\ell+1}$. Therefore, we only have to consider the partial solutions $X_4^{i,j,\ell} \cup X_4^{i,j,\ell+1}$ and $X_5^{i,j,\ell} \cup X_5^{i,j,\ell+1}$ as otherwise at least one of $u_1^{i,j,\ell+1}$ and $u_2^{i,j,\ell+1}$ will be deleted. In both cases, the connected component of $G - X$ containing $u_1^{i,j,\ell+1}$ and $u_2^{i,j,\ell+1}$ induces a path on three vertices plus some pendant edges from the triangle edges. Hence, there is no cycle in $G - (X \cup \{\hat{r}\})$.

We are left with showing that $G - X$ contains no cycle containing the root vertex $\hat{r}$. We do so by arguing that each vertex in $G - X$ has at most one path to $\hat{r}$ in $G - X$. The neighbors of $\hat{r}$ are the vertices $b_{2,1}^{i,j,\ell}$ and $c_1^{i,j,\ell}$ for all $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$. It is sufficient to show that there is no path between any of these neighbors in $G - (X \cup \{\hat{r}\})$. By the same argument as in the previous paragraph, we only have to consider consecutive path gadgets $P^{i,j,\ell}$ and $P^{i,j,\ell+1}$. By resolving the triangle edges between the clique vertices

$v_\varphi^{i,j,\ell}$, $\varphi \in [5]$, and the output vertices $b_{\varphi,\gamma}^{i,j,\ell}$, $\varphi \in [5]$, $\gamma \in \{1,2\}$, all paths in $G - \hat{r}$ between $b_{2,1}^{i,j,\ell}$ and $c_1^{i,j,\ell}$ are intersected by X. Similarly for paths in $G - \hat{r}$ between $c_1^{i,j,\ell}$ and one of the vertices $c_1^{i,j,\ell+1}$ or $b_{2,1}^{i,j,\ell+1}$ and paths between $b_{2,1}^{i,j,\ell}$ and $b_{2,1}^{i,j,\ell+1}$.

It remains to consider paths in $G - (X \cup \{\hat{r}\})$ between $b_{2,1}^{i,j,\ell}$ and $c_1^{i,j,\ell+1}$. We distinguish based on the chosen partial solution $X_\varphi^{i,j,\ell} \cup X_\varphi^{i,j,\ell+1}$, $\varphi \in [5]$. For $\varphi \neq 3$, we see that $c_1^{i,j,\ell} \in X$. For $\varphi = 3$, we see that $b_{2,1}^{i,j,\ell} \in X$. Hence, no such path can exist and X has to be a feedback vertex set. □

We say that a vertex subset $X \subseteq V(G)$ is *canonical* with respect to the twinclass $\{u_1^{i,j,\ell}, u_2^{i,j,\ell}\}$ if $u_2^{i,j,\ell} \in X$ implies $u_1^{i,j,\ell} \in X$. Since $\{u_1^{i,j,\ell}, u_2^{i,j,\ell}\}$ is a twinclass, we can always assume that we are working with a canonical subset.

Given a subset $X \subseteq V(G) \setminus \{\hat{r}\}$ that is canonical with respect to each twinclass $\{u_1^{i,j,\ell}, u_2^{i,j,\ell}\}$, we define $\mathbf{state}_X \colon [t] \times [p] \times [m(4tp+1)] \to \{\mathbf{2}, \mathbf{1}_0, \mathbf{1}_1, \mathbf{0}_0, \mathbf{0}_1\}$ by

$$\mathbf{state}_X(i,j,\ell) = \begin{cases} \mathbf{2}, & \text{if } |X \cap \{u_1^{i,j,\ell}, u_2^{i,j,\ell}\}| = 2, \\ \mathbf{1}_0, & \text{if } X \cap \{u_1^{i,j,\ell}, u_2^{i,j,\ell}\} = \{u_1^{i,j,\ell}\} \text{ and} \\ & \quad u_2^{i,j,\ell} \text{ is not root-connected in } (P^{i,j,\ell} + \hat{r}) - X, \\ \mathbf{1}_1, & \text{if } X \cap \{u_1^{i,j,\ell}, u_2^{i,j,\ell}\} = \{u_1^{i,j,\ell}\} \text{ and} \\ & \quad u_2^{i,j,\ell} \text{ is root-connected in } (P^{i,j,\ell} + \hat{r}) - X, \\ \mathbf{0}_0, & \text{if } X \cap \{u_1^{i,j,\ell}, u_2^{i,j,\ell}\} = \emptyset \text{ and} \\ & \quad u_1^{i,j,\ell} \text{ and } u_2^{i,j,\ell} \text{ are not connected in } (P^{i,j,\ell} + \hat{r}) - X, \\ \mathbf{0}_1, & \text{if } X \cap \{u_1^{i,j,\ell}, u_2^{i,j,\ell}\} = \emptyset \text{ and} \\ & \quad u_1^{i,j,\ell} \text{ and } u_2^{i,j,\ell} \text{ are connected in } (P^{i,j,\ell} + \hat{r}) - X. \end{cases}$$

Due to the assumption that X is canonical, we see that $\mathbf{state}_X$ is well-defined. We remark that the meaning of the subscript is slightly different when one or no vertex of the twinclass is in X. We also recall the notation $\mathbf{s}^1 = \mathbf{2}$, $\mathbf{s}^2 = \mathbf{1}_0$, $\mathbf{s}^3 = \mathbf{1}_1$, $\mathbf{s}^4 = \mathbf{0}_0$, and $\mathbf{s}^5 = \mathbf{0}_1$.

Lemma 8.3.3. *If there is a feedback vertex set X of G of size $|X| \leq \mathrm{cost}_{\mathcal{P}}$, then σ is satisfiable.*

Proof. Due to Lemma 8.3.1, we immediately see that $|X| = \mathrm{cost}_{\mathcal{P}}$ and $X \cap V(H)$ has to be a minimum feedback vertex set of H for any $H \in \mathcal{P}$. So, X contains precisely one vertex of each triangle in $\mathcal{P}$ and satisfies the *packing equations* for all $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$:

- $|X \cap \{v_\varphi^{i,j,\ell} : \varphi \in [5]\}| = 4$,
- $|X \cap \{b_{\varphi,1}^{i,j,\ell}, b_{\varphi,2}^{i,j,\ell} : \varphi \in [5]\}| = 8$,
- $|X \cap \{u_1^{i,j,\ell}, u_2^{i,j,\ell}, a_1^{i,j,\ell}, a_2^{i,j,\ell}, a_3^{i,j,\ell}, a_4^{i,j,\ell}\}| = 3$.

In particular, this also implies that X cannot contain the root vertex $\hat{r}$.

Furthermore, due to the standard exchange arguments for triangle edges and arrows, we can assume for any triangle edge between u and v that X contains u or v and for any arrow $A(u,v)$ that X uses the passive solution or the active solution on $A(u,v)$. Finally, we can assume that X is canonical with respect to each twinclass $\{u_1^{i,j,\ell}, u_2^{i,j,\ell}\}$, i.e., $u_2^{i,j,\ell} \in X$ implies that $u_1^{i,j,\ell} \in X$.

We begin by studying the structure of $X \cap P^{i,j,\ell}$ for any $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$. For fixed i, j, ℓ, there is a unique $\varphi \in [5]$ such that $v_\varphi^{i,j,\ell} \notin X$ due to the packing equations.

Hence, we must have $X \cap \{b^{i,j,\ell}_{\psi,1}, b^{i,j,\ell}_{\psi,2} : \psi \in [5]\} = \{b^{i,j,\ell}_{\psi,1}, b^{i,j,\ell}_{\psi,2} : \psi \in [5] \setminus \{\varphi\}\}$ due to the packing equations and the triangle edges between $v^{i,j,\ell}_{\varphi}$ and the output vertices $\{b^{i,j,\ell}_{\psi,1}, b^{i,j,\ell}_{\psi,2} : \psi \in [5] \setminus \{\varphi\}\}$.

For the left side of a path gadget $P^{i,j,\ell}$, we claim that $v^{i,j,\ell}_{\varphi} \notin X$ implies $\mathbf{state}_X(i,j,\ell) = \mathbf{s}^{\varphi'}$ with $\varphi' \geq \varphi$. For $\varphi = 1$ there is nothing to show. One can see that $(\varphi', \varphi) \notin ([3] \times \{4,5\}) \cup (\{1\} \times \{2,3\})$ by considering the size of $X \cap \{u^{i,j,\ell}_1, u^{i,j,\ell}_2, a^{i,j,\ell}_1, a^{i,j,\ell}_2, a^{i,j,\ell}_3, a^{i,j,\ell}_4\}$ in those cases: Due to the triangle edges between the clique vertices $v^{i,j,\ell}_{\psi}$, $\psi \in [5]$ and auxiliary vertices $a^{i,j,\ell}_{\gamma}$, $\gamma \in [4]$, we see that X contains at least two auxiliary vertices if $\varphi \geq 2$ and at least three if $\varphi \geq 4$. Using the packing equations, we see that this implies $|X \cap \{u^{i,j,\ell}_1, u^{i,j,\ell}_2\}| \leq 1$ if $\varphi \geq 2$ and $X \cap \{u^{i,j,\ell}_1, u^{i,j,\ell}_2\} = \emptyset$ if $\varphi \geq 4$, but the listed cases contradict this. It remains to handle the two cases $(\varphi', \varphi) = (2,3)$ and $(\varphi', \varphi) = (4,5)$. In the first case, the triangle edges between the vertex $v^{i,j,\ell}_3$ and the vertices $a^{i,j,\ell}_3$, $a^{i,j,\ell}_4$, $c^{i,j,\ell}_0$ together with the packing equations imply that $u^{i,j,\ell}_2$, $a^{i,j,\ell}_1$, $c^{i,j,\ell}_1 \notin X$, but then $\mathbf{state}_X(i,j,\ell) = \mathbf{1}_1 = \mathbf{s}^3 \neq \mathbf{s}^{\varphi'}$ because $u^{i,j,\ell}_2$, $a^{i,j,\ell}_1$, $c^{i,j,\ell}_1$, $\hat{r}$ is a path in $(P^{i,j,\ell} + \hat{r}) - X$. In the second case, the triangle edges between $v^{i,j,\ell}_5$ and the auxiliary vertices $a^{i,j,\ell}_2$, $a^{i,j,\ell}_3$, $a^{i,j,\ell}_4$ together with the packing equations imply that $u^{i,j,\ell}_1$, $u^{i,j,\ell}_2$, $a^{i,j,\ell}_1 \notin X$ and hence $\mathbf{state}_X(i,j,\ell) = \mathbf{0}_1 = \mathbf{s}^5 \neq \mathbf{s}^{\varphi'}$. This proves the claim.

Next, we claim that for any $i \in [t]$, $j \in [p]$, and $\ell_1, \ell_2 \in [m(4tp+1)]$ with $\ell_1 < \ell_2$, that the unique $\varphi_1 \in [5]$ and $\varphi_2 \in [5]$ such that $v^{i,j,\ell_1}_{\varphi_1} \notin X$ and $v^{i,j,\ell_2}_{\varphi_2} \notin X$ satisfy $\varphi_1 \geq \varphi_2$. This essentially follows from the triangularity of the considered compatibility matrix; we proceed with the formal argument. We can assume without loss of generality that $\ell_2 = \ell_1 + 1$. By the previous arguments, we know that $b^{i,j,\ell_1}_{\varphi_1,1}, b^{i,j,\ell_1}_{\varphi_1,2} \notin X$ and $\mathbf{state}_X(i,j,\ell_2) = \mathbf{s}^{\varphi'}$ with $\varphi' \geq \varphi_2$, so we are done if we can show that $\varphi_1 \geq \varphi'$. We do so by arguing that $G - X$ contains a cycle in all other cases, thus contradicting that X is a feedback vertex set. If $\varphi_1 < \varphi'$ and $(\varphi_1, \varphi') \notin \{(2,3),(4,5)\}$, then $G[\{b^{i,j,\ell_1}_{\varphi_1,1}, b^{i,j,\ell_1}_{\varphi_1,2}, u^{i,j,\ell_1+1}_1, u^{i,j,\ell_1+1}_2\} \setminus X]$ simply contains a cycle. If $(\varphi_1, \varphi') = (2,3)$, then there is a cycle passing through the root $\hat{r}$ in $G - X$ visiting $\hat{r}$, $b^{i,j,\ell_1}_{2,1}$, u^{i,j,ℓ_1+1}_2, and then uses the path to $\hat{r}$ inside $(P^{i,j,\ell_1+1} + \hat{r}) - X$ which exists due to $\mathbf{state}_X(i,j,\ell_1+1) = \mathbf{s}^{\varphi'} = \mathbf{s}^3 = \mathbf{1}_1$. If $(\varphi_1, \varphi') = (4,5)$, then there is a cycle in $G - X$ visiting u^{i,j,ℓ_1+1}_1, $b^{i,j,\ell_1}_{4,1}$, u^{i,j,ℓ_1+1}_2, and then uses the path between u^{i,j,ℓ_1+1}_2 and u^{i,j,ℓ_1+1}_1 in $(P^{i,j,\ell_1+1} + \hat{r}) - X$ which exists due to $\mathbf{state}_X(i,j,\ell_1+1) = \mathbf{s}^{\varphi'} = \mathbf{s}^5 = \mathbf{0}_1$. This shows the claim.

We say that X *cheats* from $P^{i,j,\ell}$ to $P^{i,j,\ell+1}$ if $v^{i,j,\ell}_{\varphi_1}, v^{i,j,\ell+1}_{\varphi_2} \notin X$ with $\varphi_1 > \varphi_2$. By the previous claim, there can be at most four cheats for fixed i and j. For $\gamma \in [4tp+1]$, we define the γ-th *column region* $R^{\gamma} = [(\gamma-1)m + 1, \gamma m]$. Since there are tp paths, there is a column region R^{γ} that contains no cheats by the pigeonhole principle, i.e., for all $i \in [t]$, $j \in [p]$, $\ell_1, \ell_2 \in R^{\gamma}$, $\varphi \in [5]$, we have $v^{i,j,\ell_1}_{\varphi} \notin X$ if and only if $v^{i,j,\ell_2}_{\varphi} \notin X$. Fix this γ for the remainder of the proof.

We obtain sequences $\mathbf{h}^i = (h^i_1, \ldots, h^i_p) \in [5]^p$, $i \in [t]$, by defining $h^i_j \in [5]$ as the unique number satisfying $v^{i,j,\gamma m}_{h^i_j} \notin X$. Since R^{γ} contains no cheats, we would obtain the same sequences if we use any column $\ell \in R^{\gamma} \setminus \{\gamma m\}$ instead of column γm in the definition. We obtain a truth assignment τ^i for variable group i by setting $\tau^i = \kappa^{-1}(\mathbf{h}^i)$ if $\mathbf{h}^i \in \kappa(\{0,1\}^{\beta})$ and otherwise picking an arbitrary truth assignment.

We claim that $\tau = \tau^1 \cup \cdots \cup \tau^t$ is a satisfying assignment of σ. To prove this claim, we begin by showing for all $i \in [t]$, $\ell \in R^{\gamma}$, $\mathbf{h} \in [5]^p$, that $\hat{x}^{i,\ell,\mathbf{h}} \in X$ implies $\mathbf{h} = \mathbf{h}^i$. Suppose that $\mathbf{h} = (h_1, \ldots, h_p) \neq \mathbf{h}^i$, then there is some $j \in [p]$ with $h_j \neq h^i_j$. There is an arrow from

$v^{i,j,\ell}_{h^i_j} \notin X$ to some $x^{i,\ell,\mathbf{h}}_\gamma$, $\gamma \in [4p]$, but X uses the passive solution on this arrow and hence $x^{i,\ell,\mathbf{h}}_\gamma \notin X$ as well, otherwise the packing equation for the second triangle in the arrow would be violated. To resolve the triangle in $D^{i,\ell,\mathbf{h}}$ induced by $\{x^{i,\ell,\mathbf{h}}_\gamma, y^{i,\ell,\mathbf{h}}_1, y^{i,\ell,\mathbf{h}}_2\}$, we must have $y^{i,\ell,\mathbf{h}}_1 \in X$ or $y^{i,\ell,\mathbf{h}}_2 \in X$. Hence, we must have $\hat{x}^{i,\ell,\mathbf{h}} \notin X$ in either case, as otherwise the packing equation for the triangle induced by $\{\hat{x}^{i,\ell,\mathbf{h}}, y^{i,\ell,\mathbf{h}}_1, y^{i,\ell,\mathbf{h}}_2\}$ would be violated. This proves the subclaim.

Consider clause $C_{\ell'}$, $\ell' \in [0, m-1]$, we will argue now that τ satisfies clause $C_{\ell'}$. The clause cycle Z^ℓ with $\ell = (\gamma-1)m + \ell' + 1 \in R^\gamma$ corresponds to clause $C_{\ell'}$ and since X is a feedback vertex set, there exists some $z^\ell_\eta \in X \cap Z^\ell$, $\eta \in [q5^p]$. By construction of G, there is at most one arrow incident to z^ℓ_η. If there is no incident arrow, then z^ℓ_η is not contained in any of the subgraphs in the packing $\mathcal{P}$ and hence $z^\ell_\eta \in X$ contradicts $|X| = \mathrm{cost}_\mathcal{P}$. So, there is exactly one arrow incident to z^ℓ_η and by construction of G, this arrow comes from some $\hat{x}^{i,\ell,\mathbf{h}}$. We must have $\hat{x}^{i,\ell,\mathbf{h}} \in X$ as well, because X uses the active solution on this arrow. The previous claim implies that $\mathbf{h} = \mathbf{h}^i$. Finally, such an arrow only exists, by construction, if $\kappa^{-1}(\mathbf{h}) = \kappa^{-1}(\mathbf{h}^i) = \tau^i$ satisfies clause $C_{\ell'}$, so τ must satisfy $C_{\ell'}$ as well. In this step, we use that the definition of $\mathbf{h}^i$ is independent of the considered column in region R^γ. Since the choice of $C_{\ell'}$ was arbitrary, this shows that σ is satisfiable. □

Lemma 8.3.4. *The graph $G = G(\sigma, \beta)$ has $\text{tc-pw}(G) \leq tp + (4p+3+q)5^p + \mathcal{O}(1)$ and a path decomposition of $G^q = G/\Pi_{tc}(G)$ of this width can be constructed in polynomial time.*

Proof. By construction, all sets $\{u^{i,j,\ell}_1, u^{i,j,\ell}_2\}$, $i \in [t]$, $j \in [p]$, $\ell \in [m(4tp+1)]$, are twinclasses. Let G' be the graph obtained by contracting each of these twinclasses, denoting the resulting vertex by $u^{i,j,\ell}$, then G^q is a subgraph of G'. We will show that $\text{tc-pw}(G) = \text{pw}(G^q) \leq \text{pw}(G') \leq tp + (4p+3+q)5^p + \mathcal{O}(1)$ by giving an appropriate strategy for the mixed-search-game on G' and applying Lemma 2.4.31.

Algorithm 6: Mixed-search-strategy for G'

```
Handling of arrows: whenever a searcher is placed on the tail u of an arrow A(u,v), we
  place searchers on all vertices of A(u,v) and immediately afterwards remove the
  searchers from A(u,v) − {u,v} again;
Place searcher on r̂;
Place searchers on u^{i,j,1} for all i ∈ [t], j ∈ [p];
for ℓ ∈ [m(4tp+1)] do
    Place searchers on all vertices of the clause cycle Z^ℓ;
    for i ∈ [t] do
        for h ∈ [5]^p do
            Place searchers on all vertices of the decoding gadget D^{i,ℓ,h};
        for j ∈ [p] do
            Place searchers on all vertices of P^{i,j,ℓ} − {u_1^{i,j,ℓ}, u_2^{i,j,ℓ}};
            Remove searcher from u^{i,j,ℓ} and place it on u^{i,j,ℓ+1};
            Remove searchers on P^{i,j,ℓ} − {u_1^{i,j,ℓ}, u_2^{i,j,ℓ}};
        for h ∈ [5]^p do
            Remove searchers on D^{i,ℓ,h};
    Remove searchers on Z^ℓ;
```

The mixed-search-strategy for G' is described in Algorithm 6 and the central idea is to proceed column by column and group by group in each column. The maximum number of placed searchers occurs on line 10 and is divided into one searcher for $\hat{r}$; one searcher for each $(i,j) \in [t] \times [p]$; $q5^p$ searchers for the current Z^ℓ; $(4p+3)5^p$ searchers for all $D^{i,\ell,\mathbf{h}}$ with the current i and ℓ; $\mathcal{O}(1)$ searchers for the current $P^{i,j,\ell}$; and $\mathcal{O}(1)$ searchers to handle an arrow $A(u,v)$. Note that arrows can be handled sequentially, i.e., there will be at any point in the search-strategy at most one arrow $A(u,v)$ with searchers on $A(u,v) - \{u,v\}$. Furthermore, note that whenever we place a searcher on the tail u of an arrow $A(u,v)$, we have already placed a searcher on the head v of the arrow. □

Theorem 8.3.5. *There is no algorithm that solves* Feedback Vertex Set*, given a path decomposition of $G^q = G/\Pi_{tc}(G)$ of width k, in time $\mathcal{O}^*((5-\varepsilon)^k)$ for some $\varepsilon > 0$, unless SETH fails.*

Proof. Assume that there exists an algorithm $\mathbb{A}$ that solves Feedback Vertex Set in time $\mathcal{O}^*((5-\varepsilon)^k)$ for some $\varepsilon > 0$ given a path decomposition of $G^q = G/\Pi_{tc}(G)$ of width k. Given β, we define $\delta_1 < 1$ such that $(5-\varepsilon)^{\log_5(2)} = 2^{\delta_1}$ and δ_2 such that $(5-\varepsilon)^{1/\beta} = 2^{\delta_2}$. By picking β large enough, we can ensure that $\delta = \delta_1 + \delta_2 < 1$. We will show how to solve q-Satisfiability using $\mathbb{A}$ in time $\mathcal{O}^*(2^{\delta n})$, where n is the number of variables, for all q, thus contradicting SETH.

Given a q-Satisfiability instance σ, we construct $G = G(\sigma, \beta)$ and the path decomposition from Lemma 8.3.4 in polynomial time, note that we have $q = \mathcal{O}(1)$, $\beta = \mathcal{O}(1)$ and hence $p = \mathcal{O}(1)$. We then run $\mathbb{A}$ on G and return its answer. This is correct by Lemma 8.3.2 and Lemma 8.3.3. Due to Lemma 8.3.4, we have that $\text{tc-pw}(G) \leq tp + f(q,p)$ for some function $f(q,p)$ and hence we can bound the running time by

$$\begin{aligned}
\mathcal{O}^*\left((5-\varepsilon)^{tp+f(q,p)}\right) &\leq \mathcal{O}^*\left((5-\varepsilon)^{tp}\right) &&\leq \mathcal{O}^*\left((5-\varepsilon)^{\lceil \frac{n}{\beta} \rceil p}\right) \\
\leq \mathcal{O}^*\left((5-\varepsilon)^{\frac{n}{\beta}p}\right) &\leq \mathcal{O}^*\left((5-\varepsilon)^{\frac{n}{\beta}\lceil \log_5(2^\beta) \rceil}\right) &&\leq \mathcal{O}^*\left((5-\varepsilon)^{\frac{n}{\beta}\log_5(2^\beta)}(5-\varepsilon)^{\frac{n}{\beta}}\right) \\
\leq \mathcal{O}^*\left(2^{\delta_1 \beta \frac{n}{\beta}} 2^{\delta_2 n}\right) &\leq \mathcal{O}^*\left(2^{(\delta_1+\delta_2)n}\right) &&\leq \mathcal{O}^*\left(2^{\delta n}\right),
\end{aligned}$$

hence completing the proof. □

9 Conclusion and Future Work

In this part, we have explored the fine-grained complexity landscape of several benchmark connectivity problems relative to the dense width parameters clique-width and modular-treewidth. The algorithms are obtained by bringing the cut-and-count-technique into the dense regime and fine-tuning of the states considered in the dynamic programming algorithms by problem-specific insights. The lower bounds follow the construction principle of Lokshtanov et al. [126], where the gadget designs are guided by developing an intuitive understanding of various compatibility matrices. We proceed by discussing several directions for further research related to this topic, further research directions will be discussed in Chapter 17.

Steiner Tree Parameterized by Clique-Width. STEINER TREE is arguably the purest connectivity problem, however, we were not able to determine its precise complexity parameterized by clique-width. The presented $\mathcal{O}(4^{\mathrm{cw}(G)})$-time algorithm follows very naturally from our techniques and we see no obvious potential for reducing the number of states considered in the algorithm. Let us consider the natural compatibility matrix with respect to a join and denote the vertex states by $\mathbf{0}$, $\mathbf{1}_0$, and $\mathbf{1}_1$ as in the CONNECTED VERTEX COVER[clique-width] lower bound. As in the algorithm, we observe that the presence of vertex state $\mathbf{0}$ in a label does not impose any constraints across a join. Therefore, it is sufficient to consider the label states that do not contain $\mathbf{0}$, for which we obtain the almost-triangular compatibility matrix in Table 9.1. This matrix has rank 4, but cannot be rearranged to a triangular matrix since every row contains at least 2 ones, thus we cannot prove a tight lower bound with our approach. However, we know that the base is at least 3 due to the known lower bounds parameterized by cutwidth or pathwidth. Closing the gap between the upper and lower bound is important future work. It is conceivable that using the rank of the compatibility matrix and similar methods as Groenland et al. [95] a tight lower bound for the modulo-2 counting variant can be shown, which is also solved by the presented algorithm. At the same time, it might be possible to solve the decision/optimization variant in time $\mathcal{O}^*(3^{\mathrm{cw}(G)})$ via a different technique; in that case, STEINER TREE would behave similarly to VERTEX COVER in the sense that the base for all parameters sandwiched between cutwidth and clique-width is the same.

X_1 vs. X_2	$\{\}$	$\{\mathbf{1}_0\}$	$\{\mathbf{1}_1, \mathbf{1}_0\}$	$\{\mathbf{1}_1\}$
$\{\mathbf{1}_1\}$	1	1	1	1
$\{\mathbf{1}_1, \mathbf{1}_0\}$	0	1	1	1
$\{\mathbf{1}_0\}$	0	0	1	1
$\{\}$	1	0	0	1

Tab. 9.1.: An almost-triangular compatibility matrix for STEINER TREE[clique-width].

Feedback Vertex Set Parameterized By Clique-Width. The standard way to obtain algorithms for FEEDBACK VERTEX SET via the cut-and-count-technique is to rely on the fact that a n-vertex m-edge graph G is a forest if and only if G has at most $n - m$ connected components like in the FEEDBACK VERTEX SET[modular-treewidth] algorithm. As discussed in Section 3.1, the cut-and-count-technique can be modified so that candidates with at most some number of connected components remain. By additionally counting the vertices and edges induced by a candidate, we can implement the mentioned inequality to check that a candidate induces a forest, i.e., we are solving the dual problem INDUCED FOREST instead of FEEDBACK VERTEX SET. However, as already observed by Bergougnoux and Kanté [11], the straightforward implementation of this approach only yields an XP-algorithm: to count the number of edges induced by a candidate, we need to know how many vertices of each label belong to the candidate so that the number of edges can be updated at join-operations, but storing this information leads to an XP space requirement. This problem could be avoided in the modular-treewidth algorithm, as the modular structure allows for exchange arguments showing that we only need to consider a constant number of cases for each module. Via a modified rank-based approach that avoids counting edges, Bergougnoux and Kanté [11] obtain an $\mathcal{O}^*(15^k 2^{(\omega+1)k})$-time algorithm for FEEDBACK VERTEX SET[clique-width]. Can their ideas be adapted to the cut-and-count-technique or optimized to get closer to a tight result? By our lower bound for FEEDBACK VERTEX SET[twinclass-pathwidth], we know that the optimal base for FEEDBACK VERTEX SET[clique-width] is at least 5.

Connected Odd Cycle Transversal Parameterized By Clique-Width. For CONNECTED ODD CYCLE TRANSVERSAL[clique-width], we have provided an $\mathcal{O}^*(14^k)$-time algorithm. However, similar to STEINER TREE[clique-width], our techniques are insufficient to obtain a tight result. To construct a compatibility matrix for a CONNECTED ODD CYCLE TRANSVERSAL[clique-width] lower bound, we consider the vertex states $\mathbf{0}_1$, $\mathbf{0}_2$, $\mathbf{1}_0$, $\mathbf{1}_1$, where $\mathbf{1}_0$ and $\mathbf{1}_1$ have the same meaning as in the CONNECTED VERTEX COVER[clique-width] lower bound, the states $\mathbf{0}_1$ and $\mathbf{0}_2$ represent undeleted vertices and the subscript encodes whether the vertex is colored with the first or second color. If one constructs the compatibility matrix, which we omit here, with the resulting 15 label states, then one sees by elementary arguments that it does not contain a triangular submatrix of size 14×14. However a 12×12 triangular submatrix exists and by generalizing the path gadget for CONNECTED VERTEX COVER[clique-width] and adding a simple gadget to detect the color used on a join vertex, it should be possible to obtain a corresponding lower bound, leaving a gap of 2 between upper and lower bound.

Connected Odd Cycle Transversal Parameterized By Modular-Treewidth. For CONNECTED ODD CYCLE TRANSVERSAL[modular-treewidth], it suffices to consider homogeneous cuts by Lemma 6.1.2, which decreases the 14 states of the CONNECTED ODD CYCLE TRANSVERSAL[clique-width] algorithm down to 11. Furthermore, the sensible behaviors inside a module greatly depend on its coloring properties. In a false twinclass, i.e., a module inducing an independent set, simple exchange arguments show that we should either completely delete the twinclass or color all vertices with the same color, leading to 4 states when applying cut-and-count. In a true twinclass of size at least 3, i.e., inducing a clique, we always have to delete at least one vertex and all remaining vertices must get different

colors, thus leading to 8 states. However, 2-colorable graphs that are not 2-critical, i.e., at least two vertices must be deleted to make the graph 1-colorable, yield one additional state, i.e., 9 in total, compared to true twinclasses, since we can also choose to not delete any of its vertices. It is not difficult that this is the maximum number of states that can be assumed by any module. This investigation seems to suggest that the optimal base differs when parameterizing by modular-treewidth versus twinclass-pathwidth. Following the usual lower bound approach, one can also find appropriate triangular compatibility matrices with asymmetric states for both cases, suggesting that the optimal base of CONNECTED ODD CYCLE TRANSVERSAL[modular-treewidth] should be 9 and the optimal base of CONNECTED ODD CYCLE TRANSVERSAL[twinclass-treewidth] should be 8.

Twinclass-Cutwidth and Modular-Cutwidth. In several cases, the base of the running time improves when going from treewidth and pathwidth to cutwidth, cf. Table 4.1, therefore it is natural to ask whether the same happens when we lift cutwidth into the dense setting. We first remark that the presented reductions for STEINER TREE and CONNECTED DOMINATING SET, also reduce the modular-cutwidth case to the cutwidth case, as we reduce to the same problem on the quotient graph. If the algorithms of Bojikian et al. [24] can be augmented to also handle vertex costs, then a $\mathcal{O}^*(3^k)$-time algorithm for CONNECTED DOMINATING SET[modular-cutwidth] likely exists; for STEINER TREE the base does not decrease when going to cutwidth. For the other two problems, CONNECTED VERTEX COVER and FEEDBACK VERTEX SET, the necessity of having to consider compatibility matrices with asymmetric states in the lower bounds parameterized by modular-treewidth is an indication that improved algorithms parameterized by modular-cutwidth could exist. Indeed, when considering the compatibility matrices for the cut-and-count states, which are essentially the tables we consider at the introduce edge bags, we see that both matrices only have rank 4 over $\mathbb{Z}_2$. Using these rank bounds, a sufficient extension of the framework for coloring-like problems by Bojikian et al. [24] might be able to obtain improved algorithms.

Other Width Parameters. We have studied the fine-grained complexity of connectivity problems relative to clique-width and modular-treewidth. Beyond the open questions left for these parameters, it is intriguing to extend this study to further width parameters. The parameters treewidth and cutwidth are essentially dealt with already due to Cygan et al. [51] and Bojikian et al. [24], respectively. Bergougnoux et al. [12] prove a meta-theorem yielding algorithms for connectivity problems relative to several dense width parameters; the forms of the obtained running times are $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$, $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(1)}$, $n^{\mathcal{O}(k)}$, where k is the boolean-width[1], rank-width, or mim-width, respectively. Based on the form of the running time, a natural next target would be boolean-width as there we have the biggest likelihood that our techniques transfer. Since boolean-width is smaller than clique-width, see Vatshelle [172], such results would nicely extend the fine-grained complexity landscape. For rank-width and mim-width, we know that in many cases these running time forms are essentially optimal under ETH due to lower bounds of Bergougnoux et al. [14] and Bakkane and Jaffke [6]. Can the exponent $\mathcal{O}(k^2)$ or $\mathcal{O}(k)$ also be more precisely understood in these cases by proving

[1]However, the running time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ is only obtained for a smaller class of connectivity problems parameterized by boolean-width.

lower bounds under SETH or some other hypothesis? Another interesting candidate would be twin-width, where we are currently even lacking any optimized algorithms for connectivity problems.

Hamiltonian Cycle. As discussed in Section 4.4, HAMILTONIAN CYCLE is also a widely studied connectivity problem. The algorithm of Cygan, Kratsch, and Nederlof [48] for HAMILTONIAN CYCLE[pathwidth] implies that HAMILTONIAN CYCLE[cutwidth] can also be solved in time $\mathcal{O}^*((2+\sqrt{2})^k)$, however it is unknown whether this running time can be improved for cutwidth. In the dense setting, Fomin et al. [73] have shown that no FPT-algorithm for HAMILTONIAN CYCLE[clique-width] can exist under the assumption FPT $\neq$ $W[1]$, but we do not know the complexity of HAMILTONIAN CYCLE[modular-treewidth] or HAMILTONIAN CYCLE[modular-pathwidth]. Does it remain W[1]-hard or can we find an FPT-algorithm for this case?

Part III

Beyond Width Parameters

10 Introduction

So far, our investigation has focused on the impact of various width parameters on problem complexity. Phrased informally, given a *hierarchical decomposition scheme*, cf. Chapter 1, we have only considered the effect of different separator structures, but not limits on how the different separators in such a scheme are allowed to *interact*. This part deals with such issues in the form of constraining the *depth* of the considered hierarchical decomposition schemes. That means the considered schemes only allow for a bounded number of decomposition rounds, taking into account the complexity of the separator used in each round, which yields so-called *depth parameters*. In the extreme case, we only allow a single round of decomposition, thus obtaining a single *modulator* to some simple graph class. As the guarantee of bounded depth, reduces the expressive strength of such hierarchical decomposition schemes and in particular avoids lower bounds in the style of Lokshtanov et al. [126], cf. Section 3.2.1, it is natural to ponder what algorithmic gain can be obtained by such a restriction. We consider two dimensions of algorithmic gain: first, a reduction of the *space requirement* and secondly, reducing the *running time* of the algorithms.

Since width parameters naturally lead to *dynamic programming* algorithms, which use additional space to reduce the running time, it is not surprising that the *time-optimal* algorithms parameterized by width parameters presented in Part II have large space requirements, in fact, the space requirement essentially equals the running time and is single-exponential in the parameter value. However, as already discussed by Woeginger [175], this greatly limits their applicability, because such algorithms tend to run out of available space much before the running time becomes infeasibly large. In contrast, depth parameters allow for *branching algorithms*, where we only need to store a very limited amount of intermediate results, thus yielding a small space footprint. We continue the study of connectivity problems from Part II and investigate this phenomenon and provide branching algorithms using only polynomial space for connectivity parameterized by *treedepth*, the depth analog of treewidth, in the first large section of this part.

Depth parameters allow us to shift the algorithmic paradigm from dynamic programming to branching, thus enabling greater algorithmic flexibility to tackle problems. As additionally, the standard lower bound approach of Lokshtanov et al. [126] does not apply to depth parameters, we are led to believe that depth-parameterizations yield smaller running time bases than the corresponding width-parameterization. In the remainder of this part, we provide new examples where this belief is false, even when going to the extreme case of modulator-parameterizations, by showing that several tight lower bounds relative to width parameters already hold relative to more restrictive modulator-parameterizations. While the lack of algorithmic improvements is unfortunate, such results help us understand from which *structural features* the problem complexity derives; in our case, we will see that already a single large separator is sufficient to cause the problem complexity as opposed to needing

many large disjoint separators as appearing in the lower bound constructions for width parameters.

We proceed by discussing the obtained results in more detail. The results on branching algorithms for connectivity problems parameterized by treedepth are based on joint work with Stefan Kratsch [99] and the results on tight lower bounds relative to modular-parameterizations are based on joint work with Stefan Kratsch [102] as well.

10.1 Connectivity Problems Parameterized By Treedepth

Problems that only involve local constraints usually admit single-exponential dynamic programming algorithms when parameterized by treewidth [2, 47, 163, 164, 170]. However, already for such locally checkable problems, there is evidence that the exponential space requirement of the dynamic programming algorithms is unavoidable while maintaining single-exponential running time: Pilipczuk and Wrochna [155] show that a single-exponential time and subexponential space algorithm for 3-COLORING[pathwidth] would lead to a breakthrough for the space complexity of algorithms for LONGEST COMMON SUBSEQUENCE. Drucker et al. [58] show, under plausible complexity-theoretic assumptions, that a single-exponential time and subexponential space algorithm for INDEPENDENT SET[pathwidth] does not exist. Furthermore, Chen et al. [37] devised a model for single-pass dynamic programming algorithms on tree decompositions and showed that such algorithms require exponential space for VERTEX COVER and 3-COLORING.

Faced with these obstacles when trying to reduce the space consumption for treewidth-parameterization, we turn to the parameter treedepth, which is slightly larger than pathwidth and treewidth, cf. Lemma 2.4.17, and hence the running time bases for single-exponential algorithms parameterized by treedepth are at least as small as those for treewidth. However, treedepth has been successfully used several times to obtain branching algorithms maintaining the base of the running time for treewidth, but only using polynomial space [37, 80, 155]. Obtaining such algorithms can be challenging even for problems with only local constraints, as a problem formulation has to be devised that admits recurrences of a specific form, where solutions to different subproblems can be combined via an essentially pointwise accumulation. As an example, the naive formulation of DOMINATING SET does not have recurrences of this specific form, but can be transformed appropriately by using e.g. the technique of inclusion-exclusion branching as was shown by Pilipczuk and Wrochna [155], who obtain an $\mathcal{O}^*(3^k)$-time and polynomial space algorithm for DOMINATING SET[treedepth].

Such transforms can be obtained in several cases via the algebraization technique of Lokshtanov and Nederlof [129], which has been adapted to treedepth by Fürer and Yu [80] to obtain space-efficient algorithms for e.g. counting perfect matchings. Belbasi and Fürer [8] build upon these results and present a polynomial-space algorithm for counting Hamiltonian cycles parameterized by treedepth, however its running time is not single-exponential in the treedepth. Beyond this result on Hamiltonian cycles, little was known about problems with global constraints parameterized by treedepth.

Filling this gap, we consider vertex-based connectivity problems parameterized by treedepth systematically. A priori, it seems very challenging to achieve the kind of re-

currences required for a branching algorithm in the presence of a connectivity constraint. Fortunately, the cut-and-count-technique of Cygan et al. [51] is *exactly* the right ingredient for this task: not only is the global connectivity constraint transformed into locally checkable constraints, but these local constraints are already in an appropriate form for branching algorithms.

We show that for several connectivity problems, the associated problem implied by the Cut&Count technique can be solved in single-exponential time $\mathcal{O}^*(\alpha^k)$ and polynomial space, where α is a constant and k is the depth of a given treedepth decomposition. Furthermore, the base α matches the base in the running time of the corresponding treewidth-algorithm.

Theorem 10.1.1. *There are one-sided-error Monte Carlo algorithms that, given a treedepth decomposition of depth k for a graph G, solve*

- STEINER TREE *in time $\mathcal{O}^*(3^k)$ and polynomial space,*
- CONNECTED VERTEX COVER *in time $\mathcal{O}^*(3^k)$ and polynomial space,*
- CONNECTED ODD CYCLE TRANSVERSAL *in time $\mathcal{O}^*(4^k)$ and polynomial space,*
- CONNECTED DOMINATING SET *in time $\mathcal{O}^*(4^k)$ and polynomial space,*
- FEEDBACK VERTEX SET *in time $\mathcal{O}^*(3^k)$ and polynomial space,*
- *for all $q \geq 3$,* CONNECTED DELETION TO q-COLORABLE *in time $\mathcal{O}^*((q+2)^k)$ and polynomial space.*

Related Work. Almost all other work related to the cut-and-count-technique has already been surveyed in Section 3.1 and Section 4.4, and will hence not be repeated here. Very shortly after the publishing of the original paper [99], Nederlof et al. [137] show how also edge-based connectivity problems, such as HAMILTONIAN CYCLE[treedepth] and LONG PATH[treedepth], can be solved in time $\mathcal{O}^*(5^k)$ and polynomial space. Nederlof et al. also make use of the cut-and-count-technique and reformulate their problems by counting perfect matchings in an auxiliary graph, which can be counted in polynomial space using inclusion-exclusion branching. We note however that these additional tricks come at the price of increasing the base of the running time compared to treewidth. Beyond the already mentioned branching algorithms parameterized by treedepth, Pilipczuk and Siebertz [154] provide a $2^{\mathcal{O}(h \log k)} n^{\mathcal{O}(1)}$-time and polynomial-space algorithm for SUBGRAPH ISOMORPHISM[treedepth], where h is the size of the pattern graph. Nadara et al. [135] show that treedepth can be computed exactly in time $2^{\mathcal{O}(k^2)} n$ and polynomial space, thus preserving the polynomial-space requirement even when a treedepth decomposition; we note that this algorithm also relies on inclusion-exclusion branching.

10.2 Tight Lower Bounds Under Modulator-Parameterizations

The standard application of the construction principle of Lokshtanov et al. [126] creates graphs containing arbitrarily long paths and is thus unsuitable for proving lower bounds parameterized by treedepth, as the treedepth grows logarithmically in the maximum length of contained paths, see e.g. Nešetřil and Ossona de Mendez [142]. Accordingly, very

few tight fine-grained lower bounds relative to depth-parameterizations or modulator-parameterizations are known. One exception is q-Coloring [109, 126] and its generalization to homomorphism problems [151], which do not include any cardinality or cost constraint, and beyond that we are only aware of isolated results on Vertex Cover [107] and Connected Vertex Cover [46] relative to modulator-parameterizations, and a folklore lower bound for Dominating Set[vertex cover]; it is notable that for all of these latter three results the optimal base in the running time is 2. The crux is that other lower bound proofs deal with more complex problems (e.g., deletion of vertices, packing of subgraphs, etc.) by copying the same (type of) partial solution over many *disjoint* separators; this addresses several obstacles but makes the approach unsuitable for modulator-parameterizations (or even for treedepth).

We show that a much broader range of problems may admit such improved lower bounds by giving new tight lower bounds for *vertex-deletion problems* such as Vertex Cover and Odd Cycle Transversal relative to modulator-parameters, in both the *sparse setting* and *dense setting*. In the sparse setting, we consider the size $\mathrm{dist}_{\mathrm{tw}\leq q}$ of modulators to treewidth q and in the dense setting we consider the size[1] $\mathrm{tcm}_{\mathrm{tw}\leq q}$ of twinclass-modulators to treewidth q, where q is a constant depending on the considered problem. Thus the resulting graphs may still contain long paths, but due to $\mathrm{td} \preceq^* \mathrm{dist}_{\mathrm{tw}\leq q}$ and $\text{tc-td} \preceq^* \mathrm{tcm}_{\mathrm{tw}\leq q}$, see Corollary 2.4.18 and Lemma 2.4.23, we do not only obtain new tight lower bounds for these modulator-parameters, but they also transfer to treedepth and its twinclass-variant by Lemma 2.4.12.[2]

Sparse Setting. Our main problem under consideration is Deletion to q-Colorable, i.e., delete as few vertices as possible so that a q-colorable graph remains, which specializes to Vertex Cover for $q = 1$ and to Odd Cycle Transversal for $q = 2$. Lokshtanov et al. [126] have shown that the optimal base in the running time is 2 for Vertex Cover[path/treewidth] and 3 for Odd Cycle Transversal[path/treewidth]. We show that both of these lower bounds and their generalization to more colors already hold relative to modulators to constant treewidth. Our main result for the sparse setting is as follows.

Theorem 10.2.1. *If there are $q \geq 2$, $\varepsilon > 0$ such that* Deletion to q-Colorable *can be solved in time $\mathcal{O}^*((q+1-\varepsilon)^{|M|})$, where M is a modulator to treewidth q, then SETH is false.*

The general construction for Deletion to q-Colorable, $q \geq 2$, does not work for the case $q = 1$, i.e., Vertex Cover, and we fill this gap by providing a simple ad-hoc construction for Vertex Cover parameterized by a modulator to pathwidth 2.

Theorem 10.2.2. *If there is an $\varepsilon > 0$ such that* Vertex Cover *can be solved in time $\mathcal{O}^*((2-\varepsilon)^{|M|})$, where M is a modulator to pathwidth 2, then SETH is false.*

For $q \geq 3$, Theorem 10.2.1 provides new tight lower bounds as a matching upper bound follows from generalizing the known algorithm for Odd Cycle Transversal[treewidth]. We

[1] Recall that the size of a twinclass-modulator is the number of contained twinclasses and not the number of vertices.

[2] In contrast, the inequality $\mathrm{td}(G) \leq (\mathrm{tw}(G)+1)\log_2|V(G)|$ from Lemma 2.4.17 is insufficient for such a lower bound transfer, as $\log_2|V(G)|$ appears in a multiplicative way instead of an additive way. In particular, a single-exponential algorithm parameterized by treedepth only gives rise to an XP-algorithm parameterized by treewidth.

emphasize that in Theorem 10.2.1 the treewidth bound q is the same as the bound q on the number of colors. The treewidth bound for $q = 2$ and the pathwidth bound in Theorem 10.2.2 cannot be improved due to improved algorithms obtained by Lokshtanov et al. [128] for VERTEX COVER and ODD CYCLE TRANSVERSAL parameterized by odd cycle transversal, which also implies improved algorithms when parameterized by feedback vertex set, i.e., a modulator to treewidth 1. Lokshtanov et al. [126] asked if the complexity of problems, other than q-COLORING (where a modulator to a single path is already sufficient [109]), relative to treewidth could already be explained with parameterization by feedback vertex set. As argued, this cannot be true for VERTEX COVER and ODD CYCLE TRANSVERSAL, so our results are essentially the next best explanation.

As mentioned, these tight lower bounds transfer to treedepth, thus yielding the first tight lower bounds relative to treedepth for vertex selection problems and partially resolving a question of Jaffke and Jansen [109] regarding the complexity relative to treedepth for problems studied by Lokshtanov et al. [126].

Corollary 10.2.3. *If there is a $q \geq 1$ and an $\varepsilon > 0$ such that* DELETION TO q-COLORABLE *can be solved in time $\mathcal{O}^*((q + 1 - \varepsilon)^{\mathrm{td}(G)})$, then SETH is false.*

Via simple reductions from VERTEX COVER, Theorem 10.2.2 also implies the following tight lower bounds for MAXIMUM CUT and K_q-FREE DELETION relative to modulator-parameterizations.

Theorem 10.2.4. *Assuming the SETH, the following lower bounds hold:*

- MAXIMUM CUT *cannot be solved in time $\mathcal{O}^*((2 - \varepsilon)^{|M|})$ for any $\varepsilon > 0$, where M is a modulator to treewidth at most 2.*
- *K_q-FREE DELETION cannot be solved in time $\mathcal{O}^*((2-\varepsilon)^{|M|})$ for any $\varepsilon > 0$ and $q \geq 3$, where M is a modulator to treewidth at most $q - 1$.*

The lower bound for MAXIMUM CUT also partially answers a question of Jaffke and Jansen [109], by being another problem considered by Lokshtanov et al. [126] whose running time cannot be improved when parameterizing by treedepth instead of treewidth.

Dense Setting. By considering twinclasses, which are arguably the simplest form of dense structure, and lifting the parameterizations from the sparse setting to twinclass-modulators and twinclass-treedepth, we show that analogous lower bounds can be proved in the dense setting. In particular, we obtain the first tight fine-grained lower bounds relative to a dense modulator. For the modulator-parameterization, it is not necessary to change the structural requirement of the remaining graph, i.e., graphs of treewidth q still suffice. Our main result in the dense setting is as follows.

Theorem 10.2.5. *If there are $q \geq 2$, $\varepsilon > 0$ such that* DELETION TO q-COLORABLE *can be solved in time $\mathcal{O}^*((2^q - \varepsilon)^{|\mathcal{M}|})$, where $\mathcal{M}$ is a TCM to treewidth q, then SETH is false.*

Additionally, it follows that if there are $q \geq 2$, $\varepsilon > 0$ such that DELETION TO q-COLORABLE *can be solved in time $\mathcal{O}^*((2^q - \varepsilon)^{\mathrm{tc\text{-}td}(G)})$, then SETH is false.*

Due to the results of Section 2.4.4, we have that $\mathrm{mcw} \preceq \mathrm{tc\text{-}td}$, so the lower bounds for twinclass-treedepth transfer to multi-clique-width and all parameters sandwiched between

them. Thus, the following result, relying on standard techniques for dynamic programming on graph decompositions such as the $(\min, +)$-*cover product*, yields a tight upper bound complementing the previous lower bounds.

Theorem 10.2.6. *Given a k-multi-clique-expression μ for G,* DELETION TO q-COLORABLE *on G can be solved in time $\mathcal{O}^*((2^q)^k)$.*

There is no further lower bound result for VERTEX COVER, since $q + 1 = 2^q$ for $q = 1$ and hence Theorem 10.2.2 already yields a tight lower bound for the multi-clique-width-parameterization. We remark that Jacob et al. [108] have simultaneously proven a similar upper and lower bound for the special case of ODD CYCLE TRANSVERSAL, $q = 2$, parameterized by clique-width. Their construction also proves the lower bound for linear-clique-width, but not for the more restrictive twinclass-treedepth or twinclass-modulator like our construction.

Going into more detail, the twinclasses of the modulator in the construction for Theorem 10.2.5 are *true twinclasses*, i.e., each twinclass induces a clique, and moreover they are of size q (with a small exception). Intuitively, allowing for deletions, there are 2^q possible sets of at most q colors that can be assigned to a clique of size q, e.g., the empty set $\emptyset$ corresponds to deleting the clique completely. Hence, our results essentially show that it is necessary and optimal to go through all of these color sets for each twinclass in the modulator.

In contrast, consider the situation for q-COLORING where Lampis [123] has obtained tight running times of $\mathcal{O}^*\left(\binom{q}{\lfloor q/2 \rfloor}^{\text{tc-tw}(G)}\right)$ when parameterized by twinclass-treewidth and of time $\mathcal{O}^*((2^q - 2)^{\text{cw}(G)})$ when parameterized by clique-width. Whereas the complexities for q-COLORING vary between the twinclass-setting and clique-width, this is not the case for DELETION TO q-COLORABLE. The base $\binom{q}{\lfloor q/2 \rfloor}$ is due to the fact that without deletions only color sets of the same size as the considered (true) twinclass can be attained and the most sets are possible when the size is $\lfloor q/2 \rfloor$. For clique-width, a *label class* may induce more complicated graphs than cliques or independent sets and the interaction between two label classes may also be more intricate. Lampis [123] shows that the extremal cases of color sets $\emptyset$ and $[q] = \{1, \ldots, q\}$ can be handled separately, thus yielding the base $2^q - 2$ for clique-width.

Exact Structural Thresholds. Another perspective on the achieved lower bounds is that they bring us closer to *exact structural thresholds* of complexity jumps. Let us illustrate this concept for the problem ODD CYCLE TRANSVERSAL. It is known that the optimal running time base of ODD CYCLE TRANSVERSAL[treewidth] is 3 and our results in particular show that the optimal base for ODD CYCLE TRANSVERSAL[clique-width] is 4. Based on these results, we informally know that replacing a large number of vertex separators with join-based separators causes a complexity jump from 3 to 4 for ODD CYCLE TRANSVERSAL. To better understand the relation problem complexity and input structure, we want to identify *minimal* structural features that cause a complexity jump; these are what we call an *exact structural threshold*. Returning to the ODD CYCLE TRANSVERSAL example, our lower bound relative to a twinclass-modulator to treewidth 2 shows that already changing a *single* modulator to a twinclass-modulator is sufficient for this complexity jump.

More abstractly, if we know that GRAPH PROBLEM[p], where p is some graph parameter, has optimal base α in the running time, then determining exact structural thresholds requires pushing this result in two directions. First, designing algorithms for parameters q that are *more expressive* than p such that GRAPH PROBLEM[q_{max}] maintains optimal base α, essentially showing that the increase in structural complexity when going from p to q_{max} can be handled without a penalty in time complexity. Secondly, as the results in this section are doing, determining *more restrictive* parameters q_{min} than p such that the lower bound showing optimality of base α can still be proven for GRAPH PROBLEM[q_{min}], thus ideally determining *minimal* structural features sufficient for this time complexity. In this way, the isolated result for GRAPH PROBLEM[p] is turned into a larger range of input structure, captured between q_{min} and q_{max}, that admits the same tight complexity.

Related Work. We investigate the related work in more detail and begin by considering other fine-grained lower bounds relative to modulators. Jacob et al. [107] show that VERTEX COVER parameterized by a modulator to cluster graphs, where a *cluster graph* is a disjoint union of cliques, has optimal base 2 by a reduction from HITTING SET. Their result does not imply the lower bound for VERTEX COVER[treedepth] based on parameter relationships, but this can be obtained by a slight modification of their proof.[3] Cygan et al. [46] prove that CONNECTED VERTEX COVER[connected vertex cover] has optimal base 2 by a reduction from SET COVER, however their reduction fundamentally relies on the connectivity constraint and seems unlikely to apply to more general settings.

All other tight lower bounds relative to modulator-parameterizations known to us consider coloring problems or their generalization to homomorphism problems and thus do not involve any cost or cardinality constraint. Lokshtanov et al. [126] show that q-COLORING[feedback vertex set] has optimal base q. Jaffke and Jansen [109] show that a modulator to a single path is enough for this lower bound and give a dichotomy when q-COLORING[$\mathrm{dist}_{\mathcal{F}}$] admits improved algorithms for families $\mathcal{F}$ that exclude some biclique and are closed under induced subgraphs. Piecyk and Rzążewski [151] show that the tight bounds obtained by Okrasa et al. [145] for list homomorphism problems parameterized by pathwidth and treewidth already apply relative to feedback vertex set, i.e., a modulator to a forest. We do not know any tight fine-grained lower bound relative to some depth parameter that does not already apply relative to some modulator, hence it would be interesting to find a natural problem with such complexity behavior.

In addition to these modulator lower bounds, there are some other results on improving lower bounds in the sense of exact structural thresholds. Van Geffen et al. [94] improve the lower bounds for INDEPENDENT SET[pathwidth] and DOMINATING SET[pathwidth] of Lokshtanov et al. [126] and show that they already apply to cutwidth even when restricted to planar graphs. Bojikian et al. [24] in particular also improve the parameterization from pathwidth to cutwidth of the lower bounds for STEINER TREE and CONNECTED ODD CYCLE TRANSVERSAL obtained by Cygan et al. [50].

On the algorithmic side of exact structural thresholds, the study of heterogeneous parameterizations has been gaining traction [1, 33, 34, 65, 66, 67, 103, 110], yielding the

[3] Our result on VERTEX COVER[$\mathrm{dist}_{\mathrm{pw}\leq 2}$] also follows by reduction from HITTING SET and was obtained independently of the result of Jacob et al. [107], but only accepted for publication later.

notions of $\mathcal{H}$-treewidth and *$\mathcal{H}$-elimination distance*, where the latter is a generalization of treedepth. Currently, only few of these works [65, 110] contain algorithmic results that are sufficiently optimized to apply to our fine-grained setting. Jansen et al. [110] show that Vertex Cover can be solved in time $\mathcal{O}^*(2^k)$ and Odd Cycle Transversal in time $\mathcal{O}^*(3^k)$ when parameterized by bipartite-treewidth. Eiben et al. [65] show that Maximum Cut[$\mathcal{R}_w$-treewidth] can be solved in time $\mathcal{O}^*(2^k)$, where $\mathcal{R}_w$ denotes the graphs of *rank-width* at most w. Another line of work is on dense depth parameters, we will discuss these in more detail in Chapter 15.

10.3 Organization

In Chapter 11, we present the polynomial-space branching algorithms for connectivity problems parameterized by treedepth, where we first give a general overview of such branching algorithms and then consider the problems one by one. In Chapter 12, we show a simple tight lower bound for Vertex Cover parameterized by a modulator to pathwidth 2 and consider its consequences for the problems Maximum Cut and K_q-free Deletion. Afterwards in Chapter 13, we present the main tight lower bounds of this part, namely the tight lower bounds for Deletion to q-Colorable parameterized by a (twinclass-)modulator to treewidth q, where we start with an outline of these results and then present the detailed constructions. To complement these lower bounds, we provide a straightforward algorithm for Deletion to q-Colorable[multi-clique-width] in Chapter 14. We conclude this part in Chapter 15 and consider future work.

11 Branching Algorithms on Treedepth Decompositions

We begin by giving a mostly informal overview of the ingredients for the developed branching algorithms for connectivity problems parameterized by treedepth. In the sections that follow afterwards we present the algorithms for the considered connectivity problems one by one.

11.1 Overview

Notation. We slightly extend the notation for treedepth decompositions used by Pilipczuk and Wrochna [155]. For a rooted forest $\mathcal{T} = (V, E_{\mathcal{T}})$ and a node $v \in V$ we denote by $\texttt{subtree}[v]$ the set of nodes in the subtree rooted at v, including v. By $\texttt{tail}[v]$ we denote the set of all ancestors of v, including v. Furthermore, we define $\texttt{subtree}(v) = \texttt{subtree}[v] \setminus \{v\}$, $\texttt{tail}(v) = \texttt{tail}[v] \setminus \{v\}$, and $\texttt{broom}[v] = \{v\} \cup \texttt{tail}(v) \cup \texttt{subtree}(v)$. By $\texttt{child}(v)$ we denote the children of v.

Setup. For the remainder of this section, we assume that $G = (V, E)$ is a connected graph and that the given costs $\mathbf{c}(v)$, $v \in V$, are polynomially bounded in $n = |V|$. Furthermore, let $\mathcal{T}$ be a treedepth decomposition of G of depth d. Note that since G is connected, $\mathcal{T}$ only consists of a single rooted tree.

Associated Subgraphs. At a node $v \in V(\mathcal{T}) = V(G)$, the branching algorithms consider the subgraph $G[\texttt{broom}[v]]$, which contains all ancestors and descendants of v in $\mathcal{T}$, and we want to implicitly compute a set of partial solutions living in $G[\texttt{broom}[v]]$. Let us mention a few elementary properties that directly follow from the definition of a treedepth decomposition.

- We have that $N_G[v] \subseteq \texttt{broom}[v]$, so $G[\texttt{broom}[v]]$ contains all edges incident to v.
- We have that $\texttt{broom}[v] = \texttt{tail}[v] \cup \bigcup_{u \in \texttt{child}(v)} \texttt{subtree}[u]$.
- If $|\texttt{child}(v)| \geq 2$, then $\texttt{tail}[v]$ separates $\texttt{subtree}[u]$ from $\texttt{subtree}[u']$ in G for every pair $u \neq u' \in \texttt{child}(v)$. In particular, there are no edges in G between $\texttt{subtree}[u]$ and $\texttt{subtree}[u']$ for $u \neq u' \in \texttt{child}(v)$.

Overview. Having decided upon a sensible set **States** of *vertex states* for the considered problem, our branching algorithms on a treedepth decomposition $\mathcal{T}$ roughly work as follows. We proceed top-down along the treedepth decomposition, branching on the states of the encountered vertices; these *branching decisions* will be recorded in *signatures* of the form $h\colon \texttt{dom}(h) \to \mathbf{States}$ with $\texttt{dom}(h) = \texttt{tail}(v)$ or $\texttt{dom}(h) = \texttt{tail}[v]$ for some vertex v. So,

each branch is essentially determined by a root-node path in the treedepth decomposition and an assignment of states for the vertices contained in this path. Having arrived at a vertex v with signature $g\colon \texttt{tail}(v) \to \mathbf{States}$, we want to generate all partial solutions living in $G[\texttt{broom}[v]]$ that are *compatible* with the branching decisions recorded in g. To do so, we branch over the possible vertex states of v, thus extending the signature g to a signature f with $\texttt{dom}(f) = \texttt{dom}(g) \cup \{v\} = \texttt{tail}(v) \cup \{v\} = \texttt{tail}[v]$ and perform a recursive call with signature f for every child $u \in \texttt{child}(v)$ of v in $\mathcal{T}$. These recursive calls are set up so that they can be answered independently of each other, which in particular relies on the fact that $\texttt{tail}(u) = \texttt{tail}[v] = \texttt{dom}(f)$ separates $\texttt{subtree}[u]$ from $\texttt{subtree}[u']$ for all $u \neq u' \in \texttt{child}(v)$. When these recursive calls return, where the recursive call at $u \in \texttt{child}(v)$ returns partial solutions living in $G[\texttt{broom}[u]]$, the partial solutions of the different children can, due to the properties of a treedepth decomposition, be easily combined into partial solutions living in $G[\texttt{broom}[v]]$. At the root node $\hat{r}$ of the treedepth decomposition, we have that $G[\texttt{broom}[\hat{r}]] = G[\texttt{subtree}[\hat{r}]] = G$, so we do obtain genuine solutions of the problem when the algorithm finishes.

Using Formal Polynomials to Track Costs and Weights. As we will again employ the cut-and-count-technique to solve connectivity problems, we must count the number of partial solutions at each node. However, as in the dynamic programming algorithms on tree decomposition or clique-expressions, these counts must be separated based on the *size/cost* and *weight* of the considered partial solutions. Previously, we have augmented the dynamic programming tables with additional dimensions to store this information. The same could be done for the branching algorithms on treedepth decompositions, but it turns out that *formal polynomials*, which have been used by Pilipczuk and Wrochna [155] in the same context, are a particularly convenient formalism for the recurrences that tend to appear in these branching algorithms.[1]

To illustrate, we introduce the formal variables Z_C and Z_W when tracking the cost and weight of partial solutions and we define a polynomial $P \in \mathbb{Z}[Z_C, Z_W]$ such that the coefficient of the monomial $Z_C^{\overline{c}} Z_W^{\overline{w}}$ is the number of partial solutions with cost $\overline{c}$ and weight $\overline{w}$, akin to *generating functions*. If we have two such polynomials P_1 and P_2 with P_i corresponding to a set of partial solutions $\mathcal{P}_i$ living on a vertex set V_i, $i \in [2]$, where $V_1 \cap V_2 = \emptyset$, then $P_1 + P_2$ corresponds to the union $\mathcal{P}_1 \cup \mathcal{P}_2$ and $P_1 \cdot P_2$ corresponds to the cartesian product $\mathcal{P}_1 \times \mathcal{P}_2$, where the cost of $(X, Y) \in \mathcal{P}_1 \times \mathcal{P}_2$ is given by adding the cost of X and Y and similarly for the weight. Furthermore, multiplications with e.g. Z_C^a allow us to increase the cost of all partial solutions by a. In our application, the relevant sets V_1 and V_2 are not disjoint, but we only track cost and weight on the parts that are currently disjoint and integrate the remaining parts later on; this makes tracking the number of induced edges for FEEDBACK VERTEX SET a bit more intricate, but it is quite straightforward for tracking anything related to the vertices.

[1] Formal polynomials are not quite as natural for e.g. dynamic programming on tree decompositions, as often costs and weights are subtracted at join nodes to handle overcounting. One could either introduce an additional operator on the polynomial ring to handle this or modify the dynamic program to avoid such subtraction operations. Nadara et al. [135] choose the former approach in their paper on computing treedepth exactly in polynomial space.

Inclusive and Exclusive Objects. Fix a node $v \in V(\mathcal{T}) = V(G)$ of the treedepth decomposition. In the overview, we have already seen that we consider signatures of the form $f : \texttt{tail}[v] \to \mathbf{States}$ and of the form $f : \texttt{tail}(v) \to \mathbf{States}$; we call the former *inclusive v-signatures* and the latter *exclusive v-signatures*.[2] Accordingly, we call partial solutions/objects living in $G[\texttt{broom}[v]]$ that are compatible with an inclusive v-signature f *inclusive objects* and those that are compatible with an exclusive v-signature g *exclusive objects*; both types of objects will be handled by different polynomials. The benefit of this distinction is that when computing the partial solutions for $G[\texttt{broom}[v]]$ from the partial solutions for $G[\texttt{broom}[u]]$ with $u \in \texttt{child}(v)$, we can do so in two separate steps: the *inclusive case* constructs inclusive objects at v based on exclusive objects at the children $u \in \texttt{children}(v)$, and the *exclusive case* constructs exclusive objects at v from inclusive objects at v. The recurrence for the inclusive case will be the same for all considered problems and just boils down to multiplying several polynomials, but the recurrence for the exclusive case is problem-dependent and involves the mentioned branching over the vertex states.

Polynomial Space Requirement. Branching algorithms that adhere to the described approach only have a polynomial space requirement for the following reasons. First, the computed polynomials will only be of polynomial size based on our assumption about the given costs. Secondly, when computing some polynomial at a node $v \in V(\mathcal{T}) = V(G)$, we only access polynomials associated with the same node or associated with a child $u \in \texttt{child}(v)$. Thirdly, the signatures of the accessed polynomials are either the current signature or a one-vertex extension of the current signature, i.e., $f = g[v \mapsto \mathbf{s}]$, meaning that we never change the branching decisions in the current branch but only make additional branching decisions. The last two properties show that the polynomials can be computed recursively and immediately discarded after they have been used in the relevant computation. Finally, the stack of recursive calls has depth at most $\mathcal{O}(d) = \mathcal{O}(n)$, since the recursion stops at the leaf nodes of the treedepth decomposition, which are at depth at most d, and we make two recursive calls per encountered node, one for the inclusive case and one for the exclusive case.

Formulating Appropriate Recurrences. For a problem to admit a branching algorithm (with polynomial space requirement) as described in this section, we must be able to formulate appropriate recurrences that only access polynomials corresponding to one-vertex extensions of the current signature. As already mentioned by Pilipczuk and Wrochna [155], this is straightforward when the problem only has "CSP-like"[3] constraints, i.e., each vertex is assigned some color, and feasibility can be determined by checking that the colors of the endpoints of each edge satisfy some specific relation. However, for other problems such as DOMINATING SET, where the domination constraint is not CSP-like as it comes with an existential quantifier, this might require additional tricks.[4] Furthermore, note that

[2] This naming is borrowed from the paper of Nederlof et al. [137] on HAMILTONIAN CYCLE[treedepth] and related problems.
[3] CSP is the abbreviation of constraint satisfaction problem.
[4] This behavior also required the development of a fast convolution algorithm for speeding up the join node computation in the dynamic programming algorithm for DOMINATING SET[treewidth], see van Rooij et al. [164].

connectivity constraints are not CSP-like, therefore we require an appropriate transformation to obtain the desired branching algorithms for connectivity problems.

Cut-and-Count. Fortunately, the cut-and-count-technique of Cygan et al. [51] achieves just that; the connectivity constraint is transformed into a CSP-like constraint plus counting, which is easily integrated into the branching strategy. Therefore, we apply the cut-and-count-technique to obtain such branching algorithms for connectivity problems parameterized by treedepth. No special changes to the cut-and-count-technique are required for its application on treedepth decompositions. In particular, using the technique of fixed vertices or marker vertices is not required. Not using these techniques allows us to give particularly simple definitions of the considered sets of partial solutions and simplifies the recurrences by avoiding edge cases. Up to the change of not using a fixed vertex or markers, the considered solution families for the cut-and-count-technique are the same as in the treewidth algorithms of Cygan et al. [51], however we provide different algorithms to compute them.

Inclusion-Exclusion-States. As we give a branching algorithm for Connected Dominating Set[treedepth], we also have to handle its domination constraint. Like Pilipczuk and Wrochna [155], we use the technique of inclusion-exclusion-branching which transforms the domination constraint into a CSP-like constraint by considering a different set of states and applying a small inclusion-exclusion formula. We will see that this technique can be combined with the cut-and-count-technique and thus obtain the desired branching algorithm for Connected Dominating Set[treedepth].

11.2 Steiner Tree

Let G be a connected graph and assume that the costs $\mathbf{c}(v)$, $v \in V$, are polynomially bounded in $n = |V|$. We begin by giving the standard cut-and-count-formulation for Steiner Tree. For every $V' \subseteq V$, the *relaxed solutions* in V' are given by

$$\mathcal{R}(V') = \{X \subseteq V' : K \cap V' \subseteq X\}$$

and the *candidate-cut-pairs* in V' are given by

$$\mathcal{Q}(V') = \{(X, (X_L, X_R)) \in \mathcal{C}(G[V']) : X \in \mathcal{R}(V')\},$$

i.e., $\mathcal{Q}(V')$ consists of all consistently cut induced subgraphs of $G[V']$ that contain all terminals appearing in V'. The solutions are given by $\mathcal{S} = \{X \in \mathcal{R}(V) : G[X] \text{ is connected}\}$.

For the isolation lemma, the universe is simply given by $U = V$. Hence, we sample a weight function $\mathbf{w}\colon V \to [N]$ with $N = 2n$ to guarantee an error probability of less than $1/2$ in the isolation lemma. We stratify the solutions according to their cost and weight, i.e., we define $\mathcal{S}^{\overline{c},\overline{w}} = \{X \in \mathcal{S} : \mathbf{c}(X) = \overline{c}, \mathbf{w}(X) = \overline{w}\}$ for every $\overline{c} \in [0, \mathbf{c}(V)]$ and $\overline{w} \in [0, nN]$.

The possible states of a single vertex v in a candidate-cut-pair $(X, (X_L, X_R)) \in \mathcal{Q}(V')$ are given by $\mathbf{States} = \{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$. The interpretation of the state $\mathbf{0}$ is that the vertex does

not belong to the Steiner tree. The states $\mathbf{1}_L$ and $\mathbf{1}_R$ indicate that the vertex is inside the Steiner tree and the subscript denotes to which side of the consistent cut it belongs.

For the remainder of this section, fix a treedepth decomposition $\mathcal{T}$ of G of depth d. Given a vertex v, an *inclusive v-signature* is a function $f\colon \mathtt{tail}[v] \to \mathbf{States}$, and an *exclusive v-signature* is a function $g\colon \mathtt{tail}(v) \to \mathbf{States}$. A candidate-cut-pair $(X, (X_L, X_R)) \in \mathcal{Q}(\mathtt{broom}[v])$ is *compatible* with an (inclusive/exclusive) v-signature h if for all $u \in \mathtt{dom}(h)$ we have $u \notin X$ if $h(u) = \mathbf{0}$, $u \in X_L$ if $h(u) = \mathbf{1}_L$, and $u \in X_R$ if $h(u) = \mathbf{1}_R$.

Definition 11.2.1. Given a vertex v and inclusive v-signature f, the *inclusive partial objects $\mathcal{P}[v, f]$ at v respecting f* consist of all $(X, (X_L, X_R)) \in \mathcal{Q}(\mathtt{broom}[v])$ compatible with f.

Definition 11.2.2. Given a vertex v and exclusive v-signature g, the *exclusive partial objects $\mathcal{P}(v, g)$ at v respecting g* consist of all $(X, (X_L, X_R)) \in \mathcal{Q}(\mathtt{broom}[v])$ compatible with g.

The distinction between *inclusive* and *exclusive* partial objects is important when considering a node v that has multiple children in $\mathcal{T}$. If f is an inclusive v-signature and $u \in \mathtt{child}(v)$ a child of v in $\mathcal{T}$, then we can consider $\mathcal{P}[v, f]$ and $\mathcal{P}(u, f)$. If $|\mathtt{child}(v)| = 1$, then we have $\mathtt{broom}[v] = \mathtt{broom}[u]$ and $\mathcal{P}[v, f] = \mathcal{P}(u, f)$. However, if $|\mathtt{child}(v)| \geq 2$, then $\mathtt{broom}[v] = \mathtt{tail}[v] \cup \bigcup_{u \in \mathtt{child}(v)} \mathtt{subtree}[u]$ where the union over the $\mathtt{subtree}[u]$ is disjoint. So, in the latter case an inclusive partial object in $\mathcal{P}[v, f]$ gives rise to an exclusive partial object in $\mathcal{P}(u, f)$ for each $u \in \mathtt{child}(v)$. Vice versa, since $\mathtt{broom}[u] \cap \mathtt{broom}[u'] = \mathtt{tail}[v] = \mathtt{dom}(f)$ for $u \neq u' \in \mathtt{child}(v)$, we can take an exclusive partial object from $\mathcal{P}(u, f)$ for each child $u \in \mathtt{child}(v)$ and combine them to obtain an inclusive partial object from $\mathcal{P}[v, f]$.

While inclusive and exclusive partial objects at v both live in $\mathtt{broom}[v]$, for inclusive objects we track the cost and size only inside $\mathtt{subtree}(v)$, and for exclusive objects we track them inside $\mathtt{subtree}[v]$, i.e., inside the parts of $\mathtt{broom}[v]$ where the signature does not specify the state.[5] Formally, for every inclusive v-signature f, trackers $\overline{c} \in [0, \mathbf{c}(V)]$, and $\overline{w} \in [0, nN]$, the family $\mathcal{P}^{\overline{c},\overline{w}}[v, f]$ consists of all $(X, (X_L, X_R)) \in \mathcal{P}[v, f]$ with

$$\mathbf{c}(X \cap \mathtt{subtree}(v)) = \overline{c} \text{ and } \mathbf{w}(X \cap \mathtt{subtree}(v)) = \overline{w}.$$

Similarly, for every exclusive v-signature g, trackers $\overline{c} \in [0, \mathbf{c}(V)]$, and $\overline{w} \in [0, nN]$, the family $\mathcal{P}^{\overline{c},\overline{w}}(v, g)$ consists of all $(X, (X_L, X_R)) \in \mathcal{P}(v, g)$ with

$$\mathbf{c}(X \cap \mathtt{subtree}[v]) = \overline{c} \text{ and } \mathbf{w}(X \cap \mathtt{subtree}[v]) = \overline{w}.$$

Next, we prove the main correctness lemma for the cut-and-count technique applied to Steiner Tree, namely that all unconnected candidates $X \in \mathcal{R}(V)$ cancel modulo four.

Lemma 11.2.3. *Let $\hat{r}$ be the root of the treedepth decomposition $\mathcal{T}$. For every $\overline{c} \in [0, \mathbf{c}(V)]$ and $\overline{w} \in [0, nN]$, it holds that $|\mathcal{P}^{\overline{c},\overline{w}}(\hat{r}, \emptyset)| \equiv_4 2|\mathcal{S}^{\overline{c},\overline{w}}|$.*

[5] This observation also gives rise to a definition that avoids the case distinction that follows, i.e., for a given (inclusive/exclusive) v-signature h and a compatible partial object $(X, (X_L, X_R))$ at v, we track the cost and weight of the subset $X \setminus \mathtt{dom}(h)$.

Proof. At the root $\hat{r}$, we have that $\texttt{broom}[\hat{r}] = V$ and $\texttt{tail}(\hat{r}) = \emptyset$. By Lemma 3.1.1, we can compute

$$|\mathcal{P}^{\overline{c},\overline{w}}(\hat{r},\emptyset)| = |\{(X,(X_L,X_R)) \in \mathcal{Q}(V) : \mathbf{c}(X) = \overline{c}, \mathbf{w}(X) = \overline{w}\}| = \sum_{\substack{X \in \mathcal{R}(V):\\ \mathbf{c}(X)=\overline{c},\mathbf{w}(X)=\overline{w}}} 2^{\texttt{cc}(G[X])}$$

$$= \sum_{\substack{X \in \mathcal{R}(V):\\ \mathbf{c}(X)=\overline{c},\mathbf{w}(X)=\overline{w},\\ \texttt{cc}(G[X])=1}} 2^{\texttt{cc}(G[X])} + \sum_{\substack{X \in \mathcal{R}(V):\\ \mathbf{c}(X)=\overline{c},\mathbf{w}(X)=\overline{w},\\ \texttt{cc}(G[X])\geq 2}} 2^{\texttt{cc}(G[X])} = 2|\mathcal{S}^{\overline{c},\overline{w}}| + 4\sum_{\substack{X \in \mathcal{R}(V):\\ \mathbf{c}(X)=\overline{c},\mathbf{w}(X)=\overline{w},\\ \texttt{cc}(G[X])\geq 2}} 2^{\texttt{cc}(G[X])-2}$$

$$\equiv_4 2|\mathcal{S}^{\overline{c},\overline{w}}|.$$

□

We proceed by presenting the counting procedure for Steiner Tree, which will then yield the desired algorithm for Steiner Tree[treedepth] via the standard cut-and-count approach.

Lemma 11.2.4. *There is an algorithm that given a treedepth decomposition $\mathcal{T}$ of depth d and weight function $\mathbf{w}\colon V \to [N]$ computes $|\mathcal{P}^{\overline{c},\overline{w}}(\hat{r},\emptyset)|$ modulo 4 for all $\overline{c} \in [0,\mathbf{c}(V)]$, $\overline{w} \in [0,nN]$ in time $\mathcal{O}^*(3^d)$ and polynomial space.*

Proof. Our algorithm will compute a multivariate polynomial in the formal variables Z_C and Z_W, where the coefficient of $Z_C^{\overline{c}}Z_W^{\overline{w}}$ is the cardinality of $\mathcal{P}^{\overline{c},\overline{w}}[\hat{r},\emptyset]$ modulo 4, i.e., the degree of the formal variables tracks the cost and weight of candidate solutions. The algorithm follows the approach described in Section 11.1. We begin by giving the definitions of the formal polynomials associated with the inclusive and exclusive objects.

Given a vertex v and inclusive v-signature f, the *inclusive polynomial* $P[v,f] \in \mathbb{Z}_4[Z_C,Z_W]$ *at* v *respecting* f is given by $P[v,f] = \sum_{\overline{c},\overline{w}} |\mathcal{P}^{\overline{c},\overline{w}}[v,f]| Z_C^{\overline{c}} Z_W^{\overline{w}}$, where $\overline{c} \in [0,\mathbf{c}(V)]$, $\overline{w} \in [0,nN]$. Similarly, given an exclusive v-signature g, the *exclusive polynomial* $P(v,g) \in \mathbb{Z}_4[Z_C,Z_W]$ *at* v *respecting* g is given by $P(v,g) = \sum_{\overline{c},\overline{w}} |\mathcal{P}^{\overline{c},\overline{w}}(v,g)| Z_C^{\overline{c}} Z_W^{\overline{w}}$.

We proceed by presenting the recurrences to compute the inclusive and exclusive polynomials. First, we give the recurrences for the inclusive polynomial and then the recurrence for the exclusive polynomial.

- **Leaf Case:** If v is a leaf node of $\mathcal{T}$, then $\texttt{broom}[v] = \texttt{tail}[v] = \texttt{dom}(f)$ and $\mathcal{P}[v,f]$ can contain at most the candidate-cut-pair $(f^{-1}(\{\mathbf{1}_L,\mathbf{1}_R\}),(f^{-1}(\mathbf{1}_L),f^{-1}(\mathbf{1}_R)))$. We simply verify if it satisfies the constraints of $\mathcal{Q}(\texttt{tail}[v])$, yielding the equation

$$\begin{aligned} P[v,f] &= [(f^{-1}(\mathbf{1}_L),f^{-1}(\mathbf{1}_R)) \text{ is a consistent cut of } G[f^{-1}(\{\mathbf{1}_L,\mathbf{1}_R\})] \\ &\cdot\ [K \cap \texttt{tail}[v] \subseteq f^{-1}(\{\mathbf{1}_L,\mathbf{1}_R\})]. \end{aligned} \tag{11.1}$$

- **Inclusive Case:** If v is not a leaf node of $\mathcal{T}$, then we have to combine the exclusive partial objects at the children of v to obtain the inclusive partial objects at v as discussed before. Since $\texttt{subtree}(v) = \bigcup_{u \in \texttt{child}(v)} \texttt{subtree}[u]$, where the union is disjoint, the cost and weight are obtained by adding the costs and weights of the partial objects at the children, allowing us to simply multiply the exclusive polynomials:

$$P[v,f] = \prod_{u \in \texttt{child}(v)} P(u,f). \tag{11.2}$$

- **Exclusive Case:** The exclusive polynomials at v can be computed from the inclusive polynomials at v by branching over all possible states for v and multiplying by formal variables, depending on the state, to update the cost and weight. Since the accumulators now track cost and weight inside of $\texttt{subtree}[v] = \{v\} \cup \texttt{subtree}(v)$ instead of $\texttt{subtree}(v)$, only the state of v affects the accumulator update:

$$P(v,g) = P[v, g[v \mapsto \mathbf{0}]] + (P[v, g[v \mapsto \mathbf{1}_L]] + P[v, g[v \mapsto \mathbf{1}_R]]) Z_C^{\mathbf{c}(v)} Z_W^{\mathbf{w}(v)}. \tag{11.3}$$

Based on these recurrences, the two mutually recursive functions $\texttt{computeInclusive}(v, f)$ and $\texttt{computeExclusive}(v, g)$ which compute $P[v, f]$ and $P(v, g)$ are given by Algorithm 7 and Algorithm 8 respectively. The algorithm calls $\texttt{computeExclusive}(\hat{r}, \emptyset)$ to compute $P := P(\hat{r}, \emptyset)$ and the coefficients of the monomials in P yield the desired numbers $|\mathcal{P}^{\overline{c},\overline{w}}(\hat{r}, \emptyset)|$ modulo 4.

Algorithm 7: $\texttt{computeInclusive}(v, f)$

Input: treedepth decomposition $\mathcal{T}$, costs $\mathbf{c}\colon V \to \mathbb{N}_{>0}$, weights $\mathbf{w}\colon V \to [N]$, vertex $v \in V$, inclusive v-signature f

```
if v is a leaf of T then return the result of equation (11.1);
else
    P := 1;
    for u ∈ child(v) do                                   // cf. equation (11.2)
        P := P · computeExclusive(u, f);
    return P;
```

Algorithm 8: $\texttt{computeExclusive}(v, g)$

Input: treedepth decomposition $\mathcal{T}$, costs $\mathbf{c}\colon V \to \mathbb{N}_{>0}$, weights $\mathbf{w}\colon V \to [N]$, vertex $v \in V$, exclusive v-signature g

```
for s ∈ {0, 1_L, 1_R} do
    P_s := computeInclusive(v, g[v ↦ s]);
return P_0 + (P_{1_L} + P_{1_R}) Z_C^{c(v)} Z_W^{w(v)};   // cf. equation (11.3)
```

Correctness. We will now prove the correctness of equations (11.1) through (11.3). The correctness of (11.1) has already been explained. For the remaining two equations (11.2) and (11.3), we will establish bijections between the objects counted on either side of the respective equation and argue that size and weight are updated correctly; the bijections imply that the recurrences are also correct over $\mathbb{Z}$ instead of just $\mathbb{Z}_4$.

We proceed by proving the correctness of equation (11.2), which is the only equation where the proof of correctness requires the special properties of treedepth decompositions. Consider any $(X, (X_L, X_R)) \in \mathcal{P}[v, f]$ and define $X^u = X \cap \texttt{broom}[u]$, $X_L^u = X_L \cap \texttt{broom}[u]$, $X_R^u = X_R \cap \texttt{broom}[u]$ for every $u \in \texttt{child}(v)$. We have that $(X^u, (X_L^u, X_R^u)) \in \mathcal{P}(u, f)$ for every $u \in \texttt{child}(v)$, because we are restricting the Steiner tree X and consistent cut (X_L, X_R) to the induced subgraph $G[\texttt{broom}[u]]$ of $G[\texttt{broom}[v]]$. Vice versa, we claim that any combination of exclusive partial objects $(X^u, (X_L^u, X_R^u)) \in \mathcal{P}(u, f)$ for each $u \in \texttt{child}(v)$ yields an inclusive partial object $(X, (X_L, X_R)) \in \mathcal{P}[v, f]$ by setting $X = \bigcup_u X^u$, $X_L =$

$\bigcup_u X_L^u$, $X_R = \bigcup_u X_R^u$. We clearly have $K \cap \texttt{broom}[v] \subseteq X$ since $\texttt{broom}[v] = \bigcup_u \texttt{broom}[u]$. On $\texttt{tail}[v] = \texttt{dom}(f)$ all these partial objects agree, hence no edge between $\texttt{tail}[v]$ and $\texttt{subtree}(v) = \bigcup_u \texttt{subtree}[u]$ can cross the cut (X_L, X_R). Furthermore, as there are no edges in G between $\texttt{subtree}[u]$ and $\texttt{subtree}[u']$ for $u \neq u' \in \texttt{child}(v)$ by the properties of a treedepth decomposition, we have $(X, (X_L, X_R)) \in \mathcal{C}(G[\texttt{broom}[v]])$ which shows the claim. Since the sets $X^u \setminus \texttt{tail}[v]$ partition $X \setminus \texttt{tail}[v]$, we obtain the cost and weight of $X \setminus \texttt{tail}[v]$ by summing over the costs and weights of the sets $X^u \setminus \texttt{tail}[v]$ respectively. Hence, these values are updated correctly by polynomial multiplication.

It remains to prove the correctness of (11.3). Since both inclusive and exclusive partial objects at v live in $\mathcal{Q}(\texttt{broom}[v])$, we only have to account for updating the signature and the costs and weights properly. Consider any $(X, (X_L, X_R)) \in \mathcal{P}(v, g)$, we have three possible cases depending on the state of v in this partial object.

1. If $v \notin X$, then we have that $(X, (X_L, X_R)) \in \mathcal{P}[v, f]$, where $f = g[v \mapsto \mathbf{0}]$. Vice versa, any $(X, (X_L, X_R)) \in \mathcal{P}[v, f]$ must also be in $\mathcal{P}(v, g)$. Since $X \cap \texttt{subtree}[v] = X \cap \texttt{subtree}(v)$, we do not need to update the cost or size in this case.
2. If $v \in X_L \subseteq X$, then $(X, (X_L, X_R)) \in \mathcal{P}[v, f]$, where $f = g[v \mapsto \mathbf{1}_L]$ and vice versa. Since $X \cap \texttt{subtree}[v] = (X \cap \texttt{subtree}(v)) \cup \{v\}$, multiplication by $Z_C^{\mathbf{c}(v)} Z_W^{\mathbf{w}(v)}$ correctly accounts for the cost and weight change.
3. If $v \in X_R \subseteq X$, the proof is analogous to case 2.

Note that by establishing these bijections in the proofs of correctness, we have actually shown that equations (11.1) through (11.3) are also correct when working in $\mathbb{Z}$ instead of $\mathbb{Z}_4$.

Time and Space Analysis. We finish the proof by discussing the time and space requirement of the algorithm. Observe that the coefficients of our polynomials are in $\mathbb{Z}_4$ and hence can be added and multiplied in constant time. Furthermore, all considered polynomials consist of at most polynomially many monomials as the cost and weight of a partial object are polynomial in n by assumption. Therefore, we can add and multiply the polynomials in polynomial time and hence compute recurrences (11.1) through (11.3) in polynomial time. Every polynomial $P[v, f]$ and $P(v, g)$ is computed at most once, because $P[v, f]$ is only called by $P(v, g)$ where f is an extension of g, i.e., $f = g[v \mapsto \mathbf{s}]$ for some $\mathbf{s} \in \mathbf{States}$, and $P(v, g)$ is only called by $P[w, g]$ where w is the parent of v. Hence, the recurrences only make disjoint calls and no polynomial is computed more than once. For a fixed vertex v there are at most 3^d choices for f and g. Thus, the algorithm runs in time $\mathcal{O}^*(3^d)$ for treedepth decompositions of depth d. Furthermore, the algorithm requires only polynomial space, because it has a recursion depth of $2d + 1$, every recursive call needs to store at most a constant number of polynomials, and each call only requires polynomial space by the previous discussion. □

Theorem 11.2.5. *There is a Monte-Carlo algorithm that given a treedepth decomposition of depth d for a graph G solves* Steiner Tree *on G in time $\mathcal{O}^*(3^d)$ and polynomial space. The algorithm cannot give false positives and may give false negatives with probability at most 1/2.*

Proof. We sample a weight function $\mathbf{w}\colon U \to [N]$ with $N = 2n = 2|U|$ uniformly at random and run the algorithm from Lemma 11.2.4. If there are $\overline{c} \in [0, \overline{b}]$ and $\overline{w} \in [0, nN]$ such that $|\mathcal{P}^{\overline{c},\overline{w}}(\hat{r}, \emptyset)| \not\equiv_4 0$, then the algorithm returns true. Otherwise, the algorithm returns false.

Due to Lemma 11.2.3, the algorithm cannot return false positives. Hence, assume that we are given a positive instance. By Lemma 3.1.4, the weight function $\mathbf{w}$ isolates $\{X \in \mathcal{R}(V) : G[X] \text{ is connected and } \mathbf{c}(X) \leq \overline{b}\} \neq \emptyset$ with probability at least $1/2$. If successful, there is some $\overline{c} \in [0, \overline{b}]$ and $\overline{w} \in [0, nN]$ with $|\mathcal{S}^{\overline{c},\overline{w}}| = 1$, so the algorithm will return true due to Lemma 11.2.3. □

While we have considered the node-variant of Steiner Tree here, the edge-variant of Steiner Tree can be easily reduced to the node-variant by subdividing each edge once, which increases the treedepth by at most one.

11.2.1 Adapting the Algorithm to Other Problems

The high-level structure of the counting procedure for the other problems is very similar to that of the algorithm for Steiner Tree, as they all follow the description in Section 11.1. One possible difference is that we might have to consider the solutions over a more complicated universe U than just the vertex set V, this occurs when we consider Connected Odd Cycle Transversal and its generalization Connected Deletion to q-Colorable. Also, we might want to keep track of more data of the partial solutions and hence use more than just two formal variables for the polynomials, which occurs for Feedback Vertex Set. The equation for the base case (cf. equation (11.1)) and the recurrence for $P(v, g)$ (cf. equation (11.3)) are also problem-dependent, but the equation for the inclusive case (cf. equation (11.2)) stays the same.

Time and Space Analysis. The properties that we require of the polynomials and equations in the time and space analysis, namely that the equations can be evaluated in polynomial time and every polynomial is computed at most once, remain true by the same arguments as for Steiner Tree. The running time essentially follows from the number of computed polynomials, which increases when we use more states for the vertices. Again denoting the set of states by $\mathbf{States}$, we obtain a running time of $\mathcal{O}^*(|\mathbf{States}|^d)$ on treedepth decompositions of depth d. The space analysis also transfers, as we ensure that the access structure of the recurrences remains the same, so that the same arguments still apply.

11.3 Connected Deletion to q-Colorable

Let G be connected graph and assume that the costs $\mathbf{c}(v)$, $v \in V$, are polynomially bounded in $n = |V|$.

Connected Deletion to q-Colorable generalizes the problems Connected Vertex Cover ($q = 1$) and Connected Odd Cycle Transversal ($q = 2$). The algorithm for Connected Deletion to q-Colorable is mostly straightforward, but requires a slight technical adjustment. Whereas for the previous problems, it was sufficient to consider the universe $U = V$ for the isolation lemma, this will not work here. To verify that $G - X$ is q-colorable, every partial solution (X, φ) consists of a deletion set X and a q-coloring φ of the remaining graph. However, for a fixed X there might be multiple valid q-colorings φ.

Therefore, isolating only over all cheapest connected deletion sets X does not necessarily avoid unwanted cancellations modulo four. Instead, the weights for the isolation lemma will also take the q-coloring φ into account.

To apply the cut-and-count technique, we make the following definitions. For every $V' \subseteq V$, the *relaxed solutions* in V' are given by

$$\mathcal{R}(V') = \{(X, \varphi) : X \subseteq V' \text{ and } \varphi \text{ is a } q\text{-coloring of } G[V' \setminus X]\},$$

and the *candidate-cut-pairs* in V' are given by

$$\mathcal{Q}(V') = \{((X, \varphi), (X_L, X_R)) : (X, \varphi) \in \mathcal{R}(V') \text{ and } (X, (X_L, X_R)) \in \mathcal{C}(G[V'])\}.$$

The *solutions* are given by $\mathcal{S} = \{(X, \varphi) \in \mathcal{R}(V) : G[X] \text{ is connected}\}$.

For the isolation lemma, we set the universe to $U = V \times (\{\bot\} \cup [q])$, where $\bot$ is a formal symbol representing the deletion of a vertex. So, we sample a weight function $\mathbf{w} \colon U \to [N]$ with $N = 2(q+1)n$ to guarantee an error probability of less than $1/2$ in the isolation lemma. The weight of a pair (X, φ) consisting of a deletion set X and q-coloring φ is given by $\mathbf{w}(X, \varphi) = \sum_{v \in X} \mathbf{w}(v, \bot) + \sum_{v \in \mathtt{dom}(\varphi)} \mathbf{w}(v, \varphi(v))$. We define $\mathcal{S}^{\overline{c}, \overline{w}} = \{(X, \varphi) \in \mathcal{S} : \mathbf{c}(X) = \overline{c}, \mathbf{w}(X, \varphi) = \overline{w}\}$ for every $\overline{c} \in [0, \mathbf{c}(V)]$ and $\overline{w} \in [0, nN]$.

The possible states of a single vertex v in a pair $((X, \varphi), (X_L, X_R))$ are given by $\mathbf{States} = \{\bot_L, \bot_R\} \cup [q]$, where $\bot_L$ and $\bot_R$ represent deleted vertices on the left side or the right side of the cut respectively and the states in $[q]$ represent the colors of undeleted vertices.

For the remainder of this section, fix a treedepth decomposition $\mathcal{T}$ of G of depth d. Given a vertex v, an *inclusive v-signature* is a function $f \colon \mathtt{tail}[v] \to \mathbf{States}$, and an *exclusive v-signature* is a function $g \colon \mathtt{tail}(v) \to \mathbf{States}$. A candidate-cut-pair $((X, \varphi), (X_L, X_R)) \in \mathcal{Q}(\mathtt{broom}[v])$ is *compatible* with an (inclusive/exclusive) v-signature h if for all $u \in \mathtt{dom}(h)$ we have $\varphi(u) = h(u)$ if $h(u) \in [q]$, $u \in X_L$ if $h(u) = \bot_L$, and $u \in X_R$ if $h(u) = \bot_R$.

Definition 11.3.1. Given a vertex v and inclusive v-signature f, the *inclusive partial objects* $\mathcal{P}[v, f]$ *at* v *respecting* f consist of all $((X, \varphi), (X_L, X_R)) \in \mathcal{Q}(\mathtt{broom}[v])$ compatible with f. Furthermore, for trackers $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, nN]$, the subfamily $\mathcal{P}^{\overline{c}, \overline{w}}[v, f]$ consists of all $((X, \varphi), (X_L, X_R)) \in \mathcal{P}[v, f]$ with

$$\mathbf{c}(X \cap \mathtt{subtree}(v)) = \overline{c} \text{ and } \mathbf{w}(X \cap \mathtt{subtree}(v), \varphi\big|_{\mathtt{subtree}(v) \setminus X}) = \overline{w}.$$

Definition 11.3.2. Given a vertex v and exclusive v-signature g, the *exclusive partial objects* $\mathcal{P}(v, g)$ *at* v *respecting* g consist of all $((X, \varphi), (X_L, X_R)) \in \mathcal{Q}(\mathtt{broom}[v])$ compatible with g. Furthermore, for trackers $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, nN]$, the subfamily $\mathcal{P}^{\overline{c}, \overline{w}}(v, g)$ consists of all $((X, \varphi), (X_L, X_R)) \in \mathcal{P}(v, g)$ with

$$\mathbf{c}(X \cap \mathtt{subtree}[v]) = \overline{c} \text{ and } \mathbf{w}(X \cap \mathtt{subtree}[v], \varphi\big|_{\mathtt{subtree}[v] \setminus X}) = \overline{w}.$$

Lemma 11.3.3. *Let $\hat{r}$ be the root of the treedepth decomposition $\mathcal{T}$. For every $\overline{c} \in [0, \mathbf{c}(V)]$ and $\overline{w} \in [0, nN]$, it holds that $|\mathcal{P}^{\overline{c}, \overline{w}}(\hat{r}, \emptyset)| \equiv_4 2|\mathcal{S}^{\overline{c}, \overline{w}}|$.*

Proof. By Lemma 3.1.1, we can compute

$$|\mathcal{P}^{\overline{c},\overline{w}}(\hat{r},\emptyset)| = \sum_{\substack{(X,\varphi)\in\mathcal{R}(V):\\ \mathbf{c}(X)=\overline{c},\mathbf{w}(X)=\overline{w},\\ \mathtt{cc}(G[X])=1}} 2^{\mathtt{cc}(G[X])} + \sum_{\substack{(X,\varphi)\in\mathcal{R}(V):\\ \mathbf{c}(X)=\overline{c},\mathbf{w}(X)=\overline{w},\\ \mathtt{cc}(G[X])\geq 2}} 2^{\mathtt{cc}(G[X])} \equiv_4 2|\mathcal{S}^{\overline{c},\overline{w}}|. \qquad \square$$

We can now proceed with the algorithm and present the counting procedure for Connected Deletion to q-Colorable, then the desired branching algorithm for Connected Deletion to q-Colorable[treedepth] follows via the cut-and-count-technique.

Lemma 11.3.4. *If the treedepth decomposition $\mathcal{T}$ has depth d and the costs $\mathbf{c}(v)$, $v \in V$, are polynomially bounded in $n = |V|$, then we can determine $|\mathcal{P}^{\overline{c},\overline{w}}(\hat{r},\emptyset)|$ modulo 4 for all $\overline{c} \in [0,\mathbf{c}(V)]$, $\overline{w} \in [0,nN]$ in time $\mathcal{O}^*((q+2)^d)$ and polynomial space.*

Proof. As usual, given a vertex v and inclusive v-signature f, the *inclusive polynomial* $P[v,f] \in \mathbb{Z}_4[Z_C,Z_W]$ *at v respecting f* is given by $P[v,f] = \sum_{\overline{c},\overline{w}} |\mathcal{P}^{\overline{c},\overline{w}}[v,f]| Z_C^{\overline{c}} Z_W^{\overline{w}}$. Given an exclusive v-signature g, we define the *exclusive polynomial at v respecting g* by $P(v,g) = \sum_{\overline{c},\overline{w}} |\mathcal{P}^{\overline{c},\overline{w}}(v,g)| Z_C^{\overline{c}} Z_W^{\overline{w}}$. The recurrences to compute these polynomials are as follows:

- **Leaf case:** If v is a leaf node, then $\mathtt{broom}[v] = \mathtt{tail}[v] = \mathtt{dom}(f)$ and $\mathcal{P}[v,f]$ can contain at most the candidate-cut-pair $((f^{-1}(\{\bot_L,\bot_R\}), f\big|_{f^{-1}([q])}), (f^{-1}(\bot_L), f^{-1}(\bot_R)))$; we simply verify if it satisfies the constraints of $\mathcal{Q}(\mathtt{tail}[v])$, yielding the equation

$$\begin{aligned} P[v,f] = [&(f^{-1}(\bot_L), f^{-1}(\bot_R)) \text{ is a consistent cut of } G[f^{-1}(\{\bot_L,\bot_R\})] \\ &\cdot\ [f\big|_{f^{-1}([q])} \text{ is a } q\text{-coloring of } G[f^{-1}([q])]]. \end{aligned}$$

- **Inclusive case:** If v is not a leaf node, then we need to combine the partial objects at the children of v. Since partial objects coming from different children are disjoint except for possibly the parts in $\mathtt{tail}[v]$, their costs and weights simply add up corresponding to polynomial multiplication:

$$P[v,f] = \prod_{u\in\mathtt{child}(v)} P(u,f).$$

- **Exclusive case:** We branch through all possible states of v and account for their effect on the cost and weight of the partial objects:

$$P(v,g) = (P[v,g[v\mapsto\bot_L]] + P[v,g[v\mapsto\bot_R]]) Z_C^{\mathbf{c}(v)} Z_W^{\mathbf{w}(v,\bot)} + \sum_{a\in[q]} P[v,g[v\mapsto a]] Z_W^{\mathbf{w}(v,a)}.$$

The correctness of these recurrences follows the same style of arguments as before. We highlight that in the exclusive case, the weight always increases by $\mathbf{w}(v,\bot)$ or $\mathbf{w}(v,a)$, $a \in [q]$, as also vertices that are not part of the deletion set X affect the weight of the partial objects.

For the inclusive case, we highlight that a partial object $((X,\varphi),(X_L,X_R)) \in \mathcal{P}[v,f]$ induces a partial object $((X \cap \mathtt{broom}[u], \varphi\big|_{\mathtt{broom}[u]\setminus X}), (X_L \cap \mathtt{broom}[u], X_R \cap \mathtt{broom}[u])) \in \mathcal{P}(u,f)$ for any $u \in \mathtt{child}(v)$. Vice versa, choosing partial objects $((X^u,\varphi^u),(X_L^u,X_R^u)) \in \mathcal{P}(u,f)$ for all $u \in \mathtt{child}(v)$ combine to a partial object in $\mathcal{P}[v,f]$, since they agree on

$\texttt{tail}[v] = \texttt{tail}(u)$ and since there are no edges between $\texttt{subtree}[u]$ and $\texttt{subtree}[u']$ for $u \neq u' \in \texttt{child}(v)$.

The time and space bound follow from the general discussion in Section 11.2.1. □

Theorem 11.3.5. *There exists a Monte-Carlo algorithm that given a treedepth decomposition of depth d solves* Connected Deletion to q-Colorable *in time $\mathcal{O}^*((q+2)^d)$ and polynomial space. The algorithm cannot give false positives and may give false negatives with probability at most 1/2.*

Proof. We sample a weight function $\mathbf{w}\colon U \to [N]$ with $N = 2(q+1)n$ uniformly at random and run the algorithm from Lemma 11.3.4. If there are $\overline{c} \in [0, \overline{b}]$ and $\overline{w} \in [0, nN]$ such that $|\mathcal{P}^{\overline{c},\overline{w}}(\hat{r}, \emptyset)| \not\equiv_4 0$, then the algorithm returns true. Otherwise, the algorithm returns false.

Due to Lemma 11.3.3, the algorithm cannot return false positives. Hence, assume that we are given a positive instance. By Lemma 3.1.4, the weight function $\mathbf{w}$ isolates $\{(X, \varphi) \in \mathcal{R}(V) : G[X] \text{ is connected and } \mathbf{c}(X) \leq \overline{b}\} \neq \emptyset$ with probability at least $1/2$. If successful, there is some $\overline{c} \in [0, \overline{b}]$ and $\overline{w} \in [0, nN]$ with $|\mathcal{S}^{\overline{c},\overline{w}}| = 1$, so the algorithm will return true due to Lemma 11.3.3. □

Connected Vertex Cover

The case $q = 1$, i.e., Connected Vertex Cover, can also be solved by reducing to Steiner Tree. We simply observe that the reduction presented by Cygan et al. [51] only increases the treedepth by 1 and also works with vertex costs.

Lemma 11.3.6. *Let G be a graph that contains at least two edges. There is a polynomial-time algorithm that given a* Connected Vertex Cover *instance $(G, \mathbf{c}, \overline{b})$ with a treedepth decomposition $\mathcal{T}$ of depth d for G constructs an equivalent* Steiner Tree *instance $(G', \mathbf{c}', K, \overline{b}')$ with a treedepth decomposition $\mathcal{T}'$ of depth $d+1$ for G'.*

Proof. To construct the Steiner Tree instance $(G' = (V', E'), \mathbf{c}', K, \overline{b}')$ we subdivide each edge of G with a terminal, formally we replace $e = \{u, v\} \in E$ by a new vertex w_e and two edges $\{u, w_e\}, \{w_e, v\}$ and set $K = \{w_e : e \in E(G)\}$. The new costs are given by $\mathbf{c}'(v) = \mathbf{c}(v)$ for all $v \in V' \setminus K$ and $\mathbf{c}(v) = 1$ for all $v \in K$. Setting $\overline{b}' = |K| + \overline{b}$, it is easy to see that G contains a connected vertex cover of cost at most $\overline{b}$ if and only if (G', K) contains a Steiner tree of cost at most $\overline{b}' = |K| + \overline{b} = E(G) + \overline{b}$.

To obtain the treedepth decomposition $\mathcal{T}'$ of depth $d+1$ for G' first observe that $G' - K$ is an independent set. Hence, $\mathcal{T}$ is also a valid treedepth decomposition of $G' - K$. For an edge $e = \{u, v\} \in E(G)$, we can assume without loss of generality that u is a descendant of v in $\mathcal{T}$ and attach the node corresponding to w_e directly below u as a leaf. Then, the endpoints of $\{u, w_e\}$ and $\{w_e, v\}$ are in an ancestor-descendant-relationship respectively. We repeat this process for all w_e to obtain a treedepth decomposition $\mathcal{T}'$ of depth $d+1$ for G'. □

11.4 Connected Dominating Set

Let G be a connected graph and we assume that the costs $\mathbf{c}(v)$, $v \in V$, are polynomially bounded in $n = |V|$.

Obtaining a fast polynomial-space algorithm for Connected Dominating Set is interesting, because already the algorithm running in time $\mathcal{O}^*(3^{\mathrm{td}(G)})$ and polynomial space for Dominating Set by Pilipczuk and Wrochna [155] is nontrivial. Their algorithm counts dominating sets by recursively applying a small inclusion-exclusion formula. Combining inclusion-exclusion with branching has appeared in the literature before and is also called inclusion-exclusion-branching [7, 139, 140]. We do not need any further ideas to apply the cut-and-count-technique, and it turns out that applying the inclusion-exclusion approach simultaneously is not an issue.

To motivate the need for the inclusion-exclusion technique, consider a vertex v that is dominated only by a proper descendant in the treedepth decomposition. However, this is costly to check for a naive branching algorithm: we have to pick *at least one* child $u \in \texttt{child}(v)$ and require that in $\texttt{subtree}[u]$ all partial objects contain a vertex dominating v and in the other subtrees rooted at the children of v we do not impose this restriction. As this has to be done recursively, it seems difficult to implement this for a branching algorithm while maintaining the running time and only using polynomial space.

Instead, we can declare vertices as *allowed* or *forbidden*, which will turn the *existential requirement* of picking at least one dominating vertex into a *universal requirement*. The considered partial objects *may or may not* dominate allowed vertices, but *may not* dominate forbidden vertices. If we count the number a_v of partial objects where v is declared allowed and the number b_v of partial objects where v is declared forbidden, then $a_v - b_v$ is the number of partial objects where v is dominated. The crux is that allowed and forbidden impose the *same* constraints on all subtrees rooted at the children of v, hence these branches are independent and the standard branching approach works again. We remark that the subtraction $a_v - b_v$ should only be performed when all edges incident to v have been introduced by the decomposition. As $G[\texttt{broom}[v]]$ contains all edges incident to v, it makes sense to perform subtraction when branching on the states of v.

We define the objects needed for cut-and-count in the setting of Connected Dominating Set. To apply the inclusion-exclusion branching, we need more involved definitions compared to Steiner Tree. We must distinguish, based on the considered node of the treedepth decomposition, for which vertices the subtraction $a_v - b_v$ was already performed and for which it still needs to be done; the former kind of vertices must be dominated and the latter kind not necessarily. For every $V_1, V_2 \subseteq V$ with $V_1 \cap V_2 = \emptyset$, the *relaxed solutions* are given by

$$\mathcal{R}(V_1, V_2) = \{X \subseteq V_1 \cup V_2 : V_1 \subseteq N[X]\}$$

and the *candidate-cut-pairs* are given by

$$\mathcal{Q}(V_1, V_2) = \{(X, (X_L, X_R)) \in \mathcal{C}(G[V_1 \cup V_2]) : X \in \mathcal{R}(V_1, V_2)\},$$

i.e., $\mathcal{Q}(V_1, V_2)$ consists of all consistently cut induced subgraphs of $G[V_1 \cup V_2]$ that dominate all vertices in V_1. The solutions are given by $\mathcal{S} = \{X \in \mathcal{R}(V, \emptyset) : G[X] \text{ is connected}\}$.

As for STEINER TREE, we have that the universe for the isolation lemma is $U = V$. We sample a weight function $\mathbf{w}\colon V \to [N]$ with $N = 2n$ to ensure an error probability of less than $1/2$ in the isolation lemma. We define $\mathcal{S}^{\overline{c},\overline{w}} = \{X \in \mathcal{S} : \mathbf{c}(X) = \overline{c}, \mathbf{w}(X) = \overline{w}\}$ for every $\overline{c} \in [0, \mathbf{c}(V)]$ and $\overline{w} \in [0, nN]$.

The possible states of a single vertex in a candidate-cut-pair $(X, (X_L, X_R))$ are given by $\mathbf{States} = \{\mathbf{0}_A, \mathbf{0}_F, \mathbf{1}_L, \mathbf{1}_R\}$ representing allowed vertices ($\mathbf{0}_A$), forbidden vertices ($\mathbf{0}_F$), and vertices on the left ($\mathbf{1}_L$) or the right ($\mathbf{1}_R$) side of the cut of the partial dominating set.

For the remainder of this section, fix a treedepth decomposition $\mathcal{T}$ of G of depth d. Given a vertex v, an *inclusive v-signature* is a function $f\colon \mathtt{tail}[v] \to \mathbf{States}$, and an *exclusive v-signature* is a function $g\colon \mathtt{tail}(v) \to \mathbf{States}$. A candidate-cut-pair $(X, (X_L, X_R)) \in \mathcal{Q}(V_1, V_2)$ is *compatible* with an (inclusive/exclusive) v-signature h with $\mathtt{dom}(h) = V_2$ if for all $u \in \mathtt{dom}(h) = V_2$, we have that

- $h(u) = \mathbf{0}_A$ implies $u \notin X$,
- $h(u) = \mathbf{0}_F$ implies $u \notin N_G[X]$,
- $h(u) = \mathbf{1}_L$ implies $u \in X_L$,
- $h(u) = \mathbf{1}_R$ implies $u \in X_R$.

Note that if $(X, (X_L, X_R))$ is compatible with a signature h with $h(u) = \mathbf{0}_F$ for some $u \in \mathtt{dom}(h)$, then $(X, (X_L, X_R))$ is also compatible with the signature $h[u \mapsto \mathbf{0}_A]$.

Definition 11.4.1. Given a vertex v and inclusive v-signature f, the *inclusive partial objects $\mathcal{P}[v, f]$ at v wrt. f* consist of all $(X, (X_L, X_R)) \in \mathcal{Q}(\mathtt{subtree}(v), \mathtt{tail}[v])$ compatible with f. Furthermore, for trackers $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, nN]$, the subfamily $\mathcal{P}^{\overline{c},\overline{w}}[v, f]$ consists of all $(X, (X_L, X_R)) \in \mathcal{P}[v, f]$ with $\mathbf{c}(X \cap \mathtt{subtree}(v)) = \overline{c}$ and $\mathbf{w}(X \cap \mathtt{subtree}(v)) = \overline{w}$.

Definition 11.4.2. Given a vertex v and exclusive v-signature g, the *exclusive partial objects $\mathcal{P}(v, g)$ at v wrt. g* consist of all $(X, (X_L, X_R)) \in \mathcal{Q}(\mathtt{subtree}[v], \mathtt{tail}(v))$ compatible with g. Furthermore, for trackers $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, nN]$, the subfamily $\mathcal{P}^{\overline{c},\overline{w}}(v, g)$ consists of all $(X, (X_L, X_R)) \in \mathcal{P}(v, g)$ with $\mathbf{c}(X \cap \mathtt{subtree}[v]) = \overline{c}$ and $\mathbf{w}(X \cap \mathtt{subtree}[v]) = \overline{w}$.

These definitions imply that for $(X, (X_L, X_R)) \in \mathcal{P}(v, g)$ the vertex v must be dominated by X, whereas this is not necessarily the case for $(X, (X_L, X_R)) \in \mathcal{P}[v, f]$, but both kinds of partial objects live in the subgraph $G[\mathtt{broom}[v]]$.

Lemma 11.4.3. *Let $\hat{r}$ be the root of the treedepth decomposition $\mathcal{T}$. For every $\overline{c} \in [0, \mathbf{c}(V)]$ and $\overline{w} \in [0, nN]$, it holds that $|\mathcal{P}^{\overline{c},\overline{w}}(\hat{r}, \emptyset)| \equiv_4 2|\mathcal{S}^{\overline{c},\overline{w}}|$.*

Proof. At the root $\hat{r}$, we have that $\mathtt{subtree}[\hat{r}] = V$ and $\mathtt{tail}(\hat{r}) = \emptyset$. By Lemma 3.1.1, we can compute

$$|\mathcal{P}^{\overline{c},\overline{w}}(\hat{r}, \emptyset)| = \sum_{\substack{X \in \mathcal{R}(V,\emptyset)\colon \\ \mathbf{c}(X)=\overline{c}, \mathbf{w}(X)=\overline{w}, \\ \mathtt{cc}(G[X])=1}} 2^{\mathtt{cc}(G[X])} + \sum_{\substack{X \in \mathcal{R}(V,\emptyset)\colon \\ \mathbf{c}(X)=\overline{c}, \mathbf{w}(X)=\overline{w}, \\ \mathtt{cc}(G[X])\geq 2}} 2^{\mathtt{cc}(G[X])} \equiv_4 2|\mathcal{S}^{\overline{c},\overline{w}}|. \qquad \square$$

We can now proceed with the algorithm and describe the counting procedure CONNECTED DOMINATING SET, which then yields the desired algorithm for CONNECTED DOMINATING SET[treedepth] via the cut-and-count-technique as usual.

Lemma 11.4.4. *If the treedepth decomposition $\mathcal{T}$ has depth d and the costs $\mathbf{c}(v)$, $v \in V$, are polynomially bounded in $n = |V|$, then we can determine $|\mathcal{P}(\hat{r}, \emptyset)|$ modulo 4 for all $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, nN]$ in time $\mathcal{O}^*(4^d)$ and polynomial space.*

Proof. The algorithm follows the same scheme as discussed in Section 11.1 and Section 11.2. As usual, given a vertex v and inclusive v-signature f, the *inclusive polynomial* $P[v, f] \in \mathbb{Z}_4[Z_C, Z_W]$ *at v respecting f* is given by $P[v, f] = \sum_{\overline{c},\overline{w}} |\mathcal{P}^{\overline{c},\overline{w}}[v, f]| Z_C^{\overline{c}} Z_W^{\overline{w}}$. Given an exclusive v-signature g, we define the *exclusive polynomial* $P(v, g) = \sum_{\overline{c},\overline{w}} |\mathcal{P}^{\overline{c},\overline{w}}(v, g)| Z_C^{\overline{c}} Z_W^{\overline{w}}$. The coefficients of the exclusive polynomial $P(\hat{r}, \emptyset)$ at the root $\hat{r}$ will yield the desired numbers. The recurrences to compute these polynomials are as follows:

- **Leaf case:** If v is a leaf node, then $\texttt{broom}[v] = \texttt{tail}[v] = \texttt{dom}(f)$ and $\mathcal{P}[v, f]$ can contain at most the candidate-cut-pair $(f^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\}), (f^{-1}(\mathbf{1}_L), f^{-1}(\mathbf{1}_R)))$; we simply verify if it satisfies the constraints of $\mathcal{Q}(\emptyset, \texttt{tail}[v])$, namely that we have a consistent cut and no forbidden vertices are dominated, yielding the equation

$$\begin{aligned} P[v, f] &= [(f^{-1}(\mathbf{1}_L), f^{-1}(\mathbf{1}_R)) \text{ is a consistent cut of } G[f^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\})]] \\ &\cdot \ [N[f^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\})] \cap f^{-1}(\mathbf{0}_F) = \emptyset]. \end{aligned}$$

- **Inclusive case:** If v is not a leaf node, then we need to combine the partial objects at the children of v. Since partial objects coming from different children are disjoint except for possibly the parts in $\texttt{tail}[v]$, their costs and weights simply add up which corresponds to polynomial multiplication:

$$P[v, f] = \prod_{u \in \texttt{child}(v)} P(u, f).$$

- **Exclusive case:** Here, we branch through all possible states of v and also apply the inclusion-exclusion approach to ensure that the vertex v is dominated. Furthermore, we also have to account for any changes to the cost and weight, hence we obtain the recurrence:

$$\begin{aligned} P(v, g) &= (P[v, g[v \mapsto \mathbf{0}_A]] - P[v, g[v \mapsto \mathbf{0}_F]]) \\ &+ (P[v, g[v \mapsto \mathbf{1}_L]] + P[v, g[v \mapsto \mathbf{1}_R]]) Z_C^{\mathbf{c}(v)} Z_W^{\mathbf{w}(v)}. \end{aligned}$$

The correctness of the leaf case and inclusive case follows the same style of arguments as before. In the inclusive case, the fact that there are no edges between $\texttt{subtree}[u]$ and $\texttt{subtree}[u']$ for $u \neq u' \in \texttt{child}(v)$ implies that the union of consistent cuts remains a consistent cut and that vertices from different subtrees cannot dominate each other.

We proceed by proving the correctness of the exclusive case. Note that we do not only have to account for updating the signature, costs, and weights properly, but also ensure that v is dominated. Consider any $(X, (X_L, X_R)) \in \mathcal{P}(v, g)$, we have four possible cases depending on the state of v in this partial object.

1. If $v \notin N_G[X]$, then v is not dominated and we have that $(X, (X_L, X_R)) \in \mathcal{P}[v, g[v \mapsto \mathbf{0}_A]] \cap \mathcal{P}[v, g[v \mapsto \mathbf{0}_F]]$. Hence, the recurrence counts $(X, (X_L, X_R))$ once positively and once negatively which cancel each other, so that $(X, (X_L, X_R))$ is filtered out as desired.

2. If $v \notin X$ and $v \in N[X]$, then v is dominated and we have that $(X, (X_L, X_R)) \in \mathcal{P}[v, g[v \mapsto \mathbf{0}_A]] \setminus \mathcal{P}[v, g[v \mapsto \mathbf{0}_F]]$, hence the recurrence counts $(X, (X_L, X_R))$ exactly once. Vice versa, any $(X, (X_L, X_R)) \in \mathcal{P}[v, g[v \mapsto \mathbf{0}_A]] \setminus \mathcal{P}[v, g[v \mapsto \mathbf{0}_F]]$ must also be in $\mathcal{P}(v, g)$ as this implies that v is dominated. Since $X \cap \texttt{subtree}[v] = X \cap \texttt{subtree}(v)$, we do not need to update the cost or size in this case.
3. If $v \in X_L \subseteq X$, then $(X, (X_L, X_R)) \in \mathcal{P}[v, f]$, where $f = g[v \mapsto \mathbf{1}_L]$ and vice versa. Since $X \cap \texttt{subtree}[v] = (X \cap \texttt{subtree}(v)) \cup \{v\}$, multiplication by $Z_C^{\mathbf{c}(v)} Z_W^{\mathbf{w}(v)}$ correctly accounts for the cost and weight change.
4. If $v \in X_R \subseteq X$, the proof is analogous to case 3.

The running time and space bound follow from the general discussion in Section 11.2.1. □

Theorem 11.4.5. *There exists a Monte-Carlo algorithm that given a treedepth decomposition of depth d solves* Connected Dominating Set *in time $\mathcal{O}^*(4^d)$ and polynomial space. The algorithm cannot give false positives and may give false negatives with probability at most 1/2.*

Proof. We sample a weight function $\mathbf{w} \colon U \to [N]$ with $N = 2n$ uniformly at random and run the algorithm from Lemma 11.4.4. If there are $\overline{c} \in [0, \overline{b}]$ and $\overline{w} \in [0, nN]$ such that $|\mathcal{P}^{\overline{c},\overline{w}}(\hat{r}, \emptyset)| \not\equiv_4 0$, then the algorithm returns true. Otherwise, the algorithm returns false.

Due to Lemma 11.4.3, the algorithm cannot return false positives. Hence, assume that we are given a positive instance. By Lemma 3.1.4, the weight function $\mathbf{w}$ isolates $\{X \in \mathcal{R}(V, \emptyset) : G[X] \text{ is connected and } \mathbf{c}(X) \leq \overline{b}\} \neq \emptyset$ with probability at least $1/2$. If successful, there is some $\overline{c} \in [0, \overline{b}]$ and $\overline{w} \in [0, nN]$ with $|\mathcal{S}^{\overline{c},\overline{w}}| = 1$, so the algorithm will return true due to Lemma 11.4.3. □

11.5 Feedback Vertex Set

We only consider the case that the costs $\mathbf{c}(v)$, $v \in V$, are polynomially bounded in $n = |V|$.

Feedback Vertex Set differs from the other problems in that we do not have a positive connectivity requirement, but a negative connectivity requirement, i.e., we need to ensure that the remaining graph is badly connected in the sense that it contains no cycles. Cygan et al. [49] approach this via the well-known Lemma 11.5.1 and we will do the same.

Lemma 11.5.1. *A graph G with n vertices and m edges is a forest if and only if G has at most $n - m$ connected components.*

Note that any graph G with n vertices and m edges has at least $n - m$ connected components, hence it is also true that G is a forest if and only if G has *exactly* $n - m$ connected components. The application of Lemma 11.5.1 requires that we count how many vertices and edges remain after deleting a set $X \subseteq V$ from G. We use the cut-and-count technique to bound the number of connected components of $G - X$ by working modulo a higher power of two as discussed in Section 3.1.

To be consistent with the previous algorithms, it is convenient to switch to the complementary problem Induced Forest. Although we seek to maximize $\mathbf{c}(X)$ in Induced Forest, we still call $\mathbf{c}$ a *cost* function. Clearly, G contains an induced forest X of cost $\mathbf{c}(X) \geq \overline{b}$ if and only if G contains a feedback vertex set Y of cost $\mathbf{c}(Y) \leq \mathbf{c}(V) - \overline{b}$. We only consider Induced Forest for the remainder of this section.

The basic definitions for cut-and-count in the setting of INDUCED FOREST are surprisingly simple since we do not use the marker technique, cf. Section 3.1. For every $V' \subseteq V$, the *relaxed solutions* in V' are given by $\mathcal{R}(V') = \{X : X \subseteq V'\}$ and the *candidate-cut-pairs* in V' are given by

$$\mathcal{Q}(V') = \{(X, (X_L, X_R)) \in \mathcal{C}(G[V']) : X \in \mathcal{R}(V')\},$$

i.e., $\mathcal{Q}(V')$ simply consists of all consistently cut subgraphs of $G[V']$ without any further restrictions. The *solutions* are given by $\mathcal{S} = \{X \in \mathcal{R}(V) : G[X] \text{ is a forest}\}$.

The universe for the isolation lemma is $U = V$. We sample a weight function $\mathbf{w}\colon V \to [N]$ with $N = 2n$ uniformly at random to guarantee an error probability of less than $1/2$. For every $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, nN]$, $\overline{v} \in [0, n]$, $\overline{e} \in [0, m]$, we define

$$\mathcal{S}^{\overline{c},\overline{w},\overline{v},\overline{e}} = \{X \in \mathcal{S} : \mathbf{c}(X) = \overline{c}, \mathbf{w}(X) = \overline{w}, |X| = \overline{v}, |E(G[X])| = \overline{e}\}.$$

The possible states of a single vertex v in a candidate-cut-pair $(X, (X_L, X_R)) \in \mathcal{Q}(V')$ are given by $\mathbf{States} = \{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$. The interpretation of the state $\mathbf{0}$ is that the vertex does not belong to the induced forest. The states $\mathbf{1}_L$ and $\mathbf{1}_R$ indicate that the vertex is inside the induced forest and the subscript denotes to which side of the consistent cut it belongs.

For the remainder of this section, fix a treedepth decomposition $\mathcal{T}$ of G of depth d. Given a vertex v, an *inclusive v-signature* is a function $f\colon \mathtt{tail}[v] \to \mathbf{States}$, and an *exclusive v-signature* is a function $g\colon \mathtt{tail}(v) \to \mathbf{States}$. A candidate-cut-pair $(X, (X_L, X_R)) \in \mathcal{Q}(\mathtt{broom}[v])$ is *compatible* with an (inclusive/exclusive) v-signature h if for all $u \in \mathtt{dom}(h)$ we have $u \notin X$ if $h(u) = \mathbf{0}$, $u \in X_L$ if $h(u) = \mathbf{1}_L$, and $u \in X_R$ if $h(u) = \mathbf{1}_R$.

Definition 11.5.2. Given a vertex v and inclusive v-signature f, the *inclusive partial objects $\mathcal{P}[v, f]$ at v respecting f* consist of all $(X, (X_L, X_R)) \in \mathcal{Q}(\mathtt{broom}[v])$ compatible with f.

Definition 11.5.3. Given a vertex v and exclusive v-signature g, the *exclusive partial objects $\mathcal{P}(v, g)$ at v respecting g* consist of all $(X, (X_L, X_R)) \in \mathcal{Q}(\mathtt{broom}[v])$ compatible with g.

We partition the inclusive and exclusive partial objects according to their cost, weight, number of vertices, and edges. The tracking of cost, weight, and number of vertices is standard, but the tracking of the number of edges is slightly more intricate. Note that once we arrive at $\mathtt{broom}[v]$, all edges incident to v are present in the considered subgraph $G[\mathtt{broom}[v]]$. Given an inclusive or exclusive v-signature h, we will track the number of edges in $G[X]$ that are incident to vertices in $X \cap (\mathtt{broom}[v] \setminus \mathtt{dom}(h))$.

Formally, for an inclusive v-signature f, and trackers $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, nN]$, $\overline{v} \in [0, n]$, $\overline{e} \in [0, m]$, the family $\mathcal{P}^{\overline{c},\overline{w},\overline{v},\overline{e}}[v, f]$ consists of all $(X, (X_L, X_R)) \in \mathcal{P}[v, f]$ with

- $\mathbf{c}(X \cap \mathtt{subtree}(v)) = \overline{c}$, $\mathbf{w}(X \cap \mathtt{subtree}(v)) = \overline{w}$, $|X \cap \mathtt{subtree}(v)| = \overline{v}$, and
- $|E(G[X \cap \mathtt{subtree}(v)])| + |E(X \cap \mathtt{subtree}(v), X \cap \mathtt{tail}[v])| = \overline{e}$.

Similarly, for an exclusive v-signature g, and trackers $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, nN]$, $\overline{v} \in [0, n]$, $\overline{e} \in [0, m]$, the family $\mathcal{P}^{\overline{c},\overline{w},\overline{v},\overline{e}}(v, g)$ consists of all $(X, (X_L, X_R)) \in \mathcal{P}(v, g)$ with

- $\mathbf{c}(X \cap \mathtt{subtree}[v]) = \overline{c}$, $\mathbf{w}(X \cap \mathtt{subtree}[v]) = \overline{w}$, $|X \cap \mathtt{subtree}[v]| = \overline{v}$, and
- $|E(G[X \cap \mathtt{subtree}[v]])| + |E(X \cap \mathtt{subtree}[v], X \cap \mathtt{tail}(v))| = \overline{e}$.

Observe in both definitions that $X \cap \mathtt{tail}[v]$ and $X \cap \mathtt{tail}(v)$ are completely specified by f and g respectively.

Lemma 11.5.4. *Let $\hat{r}$ be the root of the treedepth decomposition $\mathcal{T}$. For every $\bar{c} \in [0, \mathbf{c}(V)]$, $\bar{w} \in [0, nN]$, $\bar{v} \in [0, n]$, $\bar{e} \in [0, \bar{v} - 1]$, it holds that $|\mathcal{P}^{\bar{c},\bar{w},\bar{v},\bar{e}}(\hat{r}, \emptyset)| \equiv_{2^{\bar{v}-\bar{e}+1}} 2^{\bar{v}-\bar{e}} |\mathcal{S}^{\bar{c},\bar{w},\bar{v},\bar{e}}|$.*

Proof. Note that for the root $\hat{r}$, it holds that $\texttt{broom}[\hat{r}] = \texttt{subtree}[\hat{r}] = V$ and $|E(G[X])| = |E(G[X \cap \texttt{subtree}[\hat{r}]])| + |E(X \cap \texttt{subtree}[\hat{r}], X \cap \texttt{tail}(\hat{r}))|$ for all $X \subseteq V$. By Lemma 3.1.1, we have that

$$\begin{aligned}|\mathcal{P}^{\bar{c},\bar{w},\bar{v},\bar{e}}(\hat{r}, \emptyset)| &= \sum_{X\colon \texttt{cc}(G[X]) \leq \bar{v}-\bar{e}} 2^{\texttt{cc}(G[X])} + \sum_{X\colon \texttt{cc}(G[X]) > \bar{v}-\bar{e}} 2^{\texttt{cc}(G[X]} \\ &\equiv_{2^{\bar{v}-\bar{e}+1}} \sum_{X\colon \texttt{cc}(G[X]) \leq \bar{v}-\bar{e}} 2^{\texttt{cc}(G[X])} = \sum_{X\colon \texttt{cc}(G[X]) = \bar{v}-\bar{e}} 2^{\texttt{cc}(G[X])} \\ \text{[Lemma 11.5.1]} &= 2^{\bar{v}-\bar{e}} |\mathcal{S}^{\bar{c},\bar{w},\bar{v},\bar{e}}|,\end{aligned}$$

where the sums range over all $X \in \mathcal{R}(V)$ with $\mathbf{c}(X) = \bar{c}$, $\mathbf{w}(X) = \bar{w}$, $|X| = \bar{v}$, and $|E(G[X])| = \bar{e}$. □

We now present the counting procedure for Feedback Vertex Set/Induced Forest, after which the algorithm for Feedback Vertex Set[treedepth] follows using standard cut-and-count arguments.

Lemma 11.5.5. *There is an algorithm that given a treedepth decomposition $\mathcal{T}$ of depth d and weight function $\mathbf{w}\colon V \to [N]$ computes $|\mathcal{P}^{\bar{c},\bar{w},\bar{v},\bar{e}}(\hat{r}, \emptyset)|$ for all $\bar{c} \in [0, \mathbf{c}(V)]$, $\bar{w} \in [0, nN]$, $\bar{v} \in [0, n]$, $\bar{e} \in [0, \bar{v} - 1]$ in time $\mathcal{O}^*(3^d)$ and polynomial space.*

Proof. The algorithm generally follows the same framework as the counting procedure for Steiner Tree. However, there are slight adjustments to the setup of the polynomials. Since we are tracking the number of vertices and edges in addition to the cost and weight, we consider polynomials in four formal variables as opposed to two. Furthermore, since the considered power of two depends on the number of vertices and edges of the associated partial objects, we consider polynomials whose coefficients are simply (non-negative) whole numbers.

Given a vertex v and an inclusive v-signature f, the *inclusive polynomials* $P[v, f] \in \mathbb{Z}[Z_C, Z_W, Z_V, Z_E]$ *at v respecting f* is given by $P[v, f] = \sum_{\bar{c},\bar{w},\bar{v},\bar{e}} |\mathcal{P}^{\bar{c},\bar{w},\bar{v},\bar{e}}[v, f]| Z_C^{\bar{c}} Z_W^{\bar{w}} Z_V^{\bar{v}} Z_E^{\bar{e}}$. Given an exclusive v-signature g, the *exclusive polynomials* $P(v, g) \in \mathbb{Z}[Z_C, Z_W, Z_V, Z_E]$ *at v respecting g* is given by $P(v, g) = \sum_{\bar{c},\bar{w},\bar{v},\bar{e}} |\mathcal{P}^{\bar{c},\bar{w},\bar{v},\bar{e}}(v, g)| Z_C^{\bar{c}} Z_W^{\bar{w}} Z_V^{\bar{v}} Z_E^{\bar{e}}$.

Note that there are at most 3^n consistently cut induced subgraphs of G, thus all occurring coefficients have size polynomial in n and can be added and multiplied in polynomial time.

We now present the recurrences used to compute the polynomials $P[v, f]$ and $P(v, g)$.

- **Leaf case:** If v is a leaf node, then $\texttt{broom}[v] = \texttt{tail}[v] = \texttt{dom}(f)$ and $\mathcal{P}[v, f]$ can contain at most the candidate-cut-pair $(f^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\}), (f^{-1}(\mathbf{1}_L), f^{-1}(\mathbf{1}_R)))$; we simply verify that it is a consistently cut subgraph, i.e.

$$P[v, f] = [(f^{-1}(\mathbf{1}_L), f^{-1}(\mathbf{1}_R)) \text{ is a consistent cut of } G[f^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\})]].$$

- **Inclusive case:** If v is not a leaf node, then we must combine the partial objects at the children of v. Since the vertex sets and edge sets whose cost, weight, and size we track

are disjoint for partial objects coming from different children, we can simply multiply the corresponding polynomials

$$P[v, f] = \prod_{u \in \texttt{child}(v)} P(u, f).$$

- **Exclusive case:** We branch through all possible states of v and account for their effect on the different accumulators of the partial objects. In particular, the number of edges increases by $|E(\{v\}, \texttt{tail}(v)) \cap E(G[X])| = |N(v) \cap g^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\})| =: \bar{e}_g$ for every $(X, (X_L, X_R)) \in \mathcal{P}(v, g)$ with $v \in X$. Hence, the recurrence is given by

$$P(v, g) = P[v, g[v \mapsto \mathbf{0}]] + (P[v, g[v \mapsto \mathbf{1}_L]] + P[v, g[v \mapsto \mathbf{1}_R]]) Z_C^{\mathbf{c}(v)} Z_W^{\mathbf{w}(v)} Z_V Z_E^{\bar{e}_g}.$$

As in the proof for STEINER TREE, we can design two mutually recursive functions based on these recurrences that allow us to compute $P[\hat{r}, \emptyset]$ as desired. The correctness of these recurrences also largely follows by very similar arguments as for STEINER TREE. We show that the number of edges is correctly updated in the inclusive and exclusive case, all other details are omitted.

For the inclusive case, we have an inclusive partial object $(X, (X_L, X_R)) \in \mathcal{P}[v, f]$ and exclusive partial objects $(X^u, (X_L^u, X_R^u)) \in \mathcal{P}(u, f)$ with $X^u = X \cap \texttt{broom}[u]$ for every $u \in \texttt{child}(v)$. Let $\bar{e} = |E(G[X \cap \texttt{subtree}(v)])| + |E(X \cap \texttt{subtree}(v), X \cap \texttt{tail}[v])|$ and $\bar{e}^u = |E(G[X^u \cap \texttt{subtree}[u]])| + |E(X^u \cap \texttt{subtree}[u], X^u \cap \texttt{tail}(u))|$ for every $u \in \texttt{child}(v)$. We claim that $\bar{e} = \sum_{u \in \texttt{child}(v)} \bar{e}^u$. We know that $X \cap \texttt{subtree}(v)$ is the disjoint union of the $X^u \cap \texttt{subtree}[u]$, $u \in \texttt{child}(v)$, and that $E(\texttt{subtree}[u], \texttt{subtree}[u']) = \emptyset$ for every $u \neq u' \in \texttt{child}(v)$, thus handling the first summand of $\bar{e}$ and every $\bar{e}^u$. Since $X^u \cap \texttt{tail}(u) = X \cap \texttt{tail}[v]$ for every $u \in \texttt{child}(v)$, also the second summands are handled, which proves the claim and shows that the recurrence for the inclusive case is correct.

For the exclusive case, we have an exclusive partial object $(X, (X_L, X_R)) \in \mathcal{P}(v, g)$ with $\bar{e} = |E(G[X \cap \texttt{subtree}[v]])| + |E(X \cap \texttt{subtree}[v], X \cap \texttt{tail}(v))|$ and there is some $\mathbf{s} \in \mathbf{States}$ with $(X, (X_L, X_R)) \in \mathcal{P}[v, g[v \mapsto \mathbf{s}]]$. Let $\bar{e}' = |E(G[X \cap \texttt{subtree}(v)])| + |E(X \cap \texttt{subtree}(v), X \cap \texttt{tail}[v])|$. If $\mathbf{s} = \mathbf{0}$, then $v \notin X$ and clearly $\bar{e} = \bar{e}'$. If $\mathbf{s} \in \{\mathbf{1}_L, \mathbf{1}_R\}$, then $v \in X$ and the following equations hold:

- $|E(G[X \cap \texttt{subtree}[v]])| = |E(G[X \cap \texttt{subtree}(v)])| + |E(\{v\}, X \cap \texttt{subtree}(v))|$,
- $|E(X \cap \texttt{subtree}[v], X \cap \texttt{tail}(v))| = |E(X \cap \texttt{subtree}(v), X \cap \texttt{tail}(v))| + |E(\{v\}, X \cap \texttt{tail}(v))|$,
- $|E(X \cap \texttt{subtree}(v), X \cap \texttt{tail}[v])| = |E(X \cap \texttt{subtree}(v), X \cap \texttt{tail}(v))| + |E(\{v\}, X \cap \texttt{subtree}(v))|$.

Therefore, $\bar{e} - \bar{e}' = |E(\{v\}, X \cap \texttt{tail}(v))| = |N(v) \cap g^{-1}(\{\mathbf{1}_L, \mathbf{1}_R\})|$, which proves the correctness of the recurrence.

The running time and space bound essentially follow from the general discussion in Section 11.2.1, but notice that the running time and space requirement are larger by a polynomial factor due to the non-constant coefficients. □

By putting everything together, we obtain the following theorem.

Theorem 11.5.6. *There is a Monte-Carlo algorithm that given a treedepth decomposition of depth d for a graph G solves* FEEDBACK VERTEX SET/INDUCED FOREST *on G in time $\mathcal{O}^*(3^d)$ and polynomial space. The algorithm cannot give false positives and may give false negatives with probability at most 1/2.*

Proof. If solving FEEDBACK VERTEX SET, we first switch to the complementary problem INDUCED FOREST as discussed before.

We sample a weight function $\mathbf{w}\colon U \to [N]$ with $N = 2n$ uniformly at random and run the algorithm from Lemma 11.5.5. If there are some $\overline{c} \in [\overline{b}, \mathbf{c}(V)]$, $\overline{w} \in [0, nN]$, $\overline{v} \in [0, n]$, $\overline{e} \in [0, \overline{v} - 1]$ such that $|\mathcal{P}^{\overline{c},\overline{w},\overline{v},\overline{e}}(\hat{r}, \emptyset)| \not\equiv_{2^{\overline{v}-\overline{e}+1}} 0$, then the algorithm returns true. Otherwise, the algorithm returns false.

Due to Lemma 11.5.4, the algorithm cannot return false positives. Hence, assume that we are given a positive instance. By Lemma 3.1.4, the weight function $\mathbf{w}$ isolates $\{X \in \mathcal{R}(V) : G[X] \text{ is a forest and } \mathbf{c}(X) \geq \overline{b}\} \neq \emptyset$ with probability at least $1/2$. If successful, there are some $\overline{c} \in [\overline{b}, \mathbf{c}(V)]$, $\overline{w} \in [0, nN]$, $\overline{v} \in [0, n]$, $\overline{e} \in [0, \overline{v} - 1]$ with $|\mathcal{S}^{\overline{c},\overline{w},\overline{v},\overline{e}}| = 1$, so the algorithm will return true due to Lemma 11.2.3. □

Theorem 11.5.6 allows us to easily reobtain a result of Cygan et al. [51] on FEEDBACK VERTEX SET[feedback vertex set]. The assumption that a feedback vertex set is given can be eliminated by using the technique of *iterative compression*. This result has been superseded by results of Li and Nederlof [125]; they present an $\mathcal{O}^*(2.7^k)$-time and exponential-space algorithm and an $\mathcal{O}^*(2.8446^k)$-time and polynomial-space algorithm for this problem.

Corollary 11.5.7 ([51])**.** *There is a Monte-Carlo algorithm that given a feedback vertex set of size s solves* FEEDBACK VERTEX SET *with unit costs in time $\mathcal{O}^*(3^s)$ and polynomial space. The algorithm cannot give false positives and may give false negatives with probability at most 1/2.*

Proof. By Corollary 2.4.18, we have that $\mathrm{td} \preceq^* \mathrm{fvs}$ constructively. Hence, we can appropriately transform the given feedback vertex set and then run the algorithm from Theorem 11.5.6. The running time follows by Lemma 2.4.12. □

12 Modulator Lower Bound for Vertex Cover

In Section 12.1 we prove the lower bound for Vertex Cover parameterized by a modulator to pathwidth 2 and in Section 12.2 we show that this lower bound also implies lower bounds for Maximum Cut and K_q-free Deletion. Note that Vertex Cover is just another name for Deletion to 1-Colorable, but the case $q = 1$ is not covered by the lower bounds in Chapter 13.

12.1 Lower Bound

This section is devoted to establishing the lower bound for Vertex Cover when parameterized by a modulator to pathwidth 2, i.e., Theorem 12.1.1. For Vertex Cover, we do not need to convert between different bases in the running time. By additionally reducing from q-Hitting Set instead of q-Satisfiability, we obtain a significantly simplified reduction that does not require the trick of Cygan et al. [46]. We construct a graph so that each vertex in the modulator corresponds to an element in the universe of the q-Hitting Set instance. We construct gadgets of pathwidth at most 2 that simulate the q-Hitting Set constraints on the modulator. Using Theorem 2.1.2, this implies the desired lower bound for Vertex Cover if SETH is true. Vertex Cover[multi-clique-width] can be solved in time $\mathcal{O}^*(2^k)$ by an algorithm of Fürer [79] and by Section 2.4.4 this implies that the obtained lower bound is tight. So, in the exceptional case of Vertex Cover, the complexity when parameterized by clique-width is already explained by the sparse setting and we do not need to consider twinclasses.

Theorem 12.1.1. *If* Vertex Cover *can be solved in time* $\mathcal{O}^*((2-\varepsilon)^{|M|})$ *for some* $\varepsilon > 0$*, where* M *is a modulator to pathwidth at most 2, then SETH is false.*

Suppose that we can solve Vertex Cover in time $\mathcal{O}^*((2-\varepsilon)^{|M|})$ for some $\varepsilon > 0$. Fix an integer q and let $(U = \{u_1, \ldots, u_n\}, \mathcal{F})$ be a q-Hitting Set instance with $|\mathcal{F}| = m$ sets of size at most q each, and budget $\overline{h}$. The elements of the j-th set in $\mathcal{F}$, $j \in [m]$, are denoted by $\{u_1^j, u_2^j, \ldots, u_{p_j}^j\}$, $p_j \leq q$. We will reduce $(U, \mathcal{F})$ to a Vertex Cover instance $G = G(U, \mathcal{F})$ with budget $\overline{b}$, which will be defined later, and which admits a modulator to pathwidth 2 of size n.

Construction. We create an independent set of n central vertices $W = \{w_1, \ldots, w_n\}$. The vertices in W that are chosen by a vertex cover will correspond to the hitting set. For the j-th set $\{u_1^j, u_2^j, \ldots, u_{p_j}^j\} \in \mathcal{F}$, $p_j \leq q$, we create a triangle path, denoted by P^j: The triangle path P^j consists of p_j vertices a_s^j, where $s \in [p_j]$, and $2p_j + 2$ vertices b_s^j, where $s \in [2p_j + 2]$,

such that $a^j_s, b^j_{2s}, b^j_{2s+1}$ form a triangle for all $s \in [p_j]$ and $b^j_1, b^j_2 \ldots, b^j_{2p_j+2}$ form a path, see Figure 12.1. We connect P^j to W by adding, for all $s \in [p_j]$, an edge between a^j_s and $w_{s'}$, where $s' \in [n]$ so that $u_{s'} = u^j_s$. The key property of the triangle path P^j is that it costs more to cover P^j if no vertex in $N(P^j) \subseteq W$ is inside the vertex cover. Since the budget constraint will be tight, this implies that a vertex cover has to take at least one vertex in each $N(P^j)$.

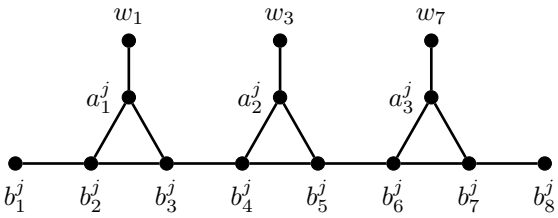


Fig. 12.1.: The triangle path P^j, where the j-th set consists of u_1, u_3, and u_7.

Lemma 12.1.2. *Let $j \in [m]$ and X be a vertex cover of G, then $|X \cap P^j| \geq 2p_j$ and:*

1. *If $X \cap N(P^j) = \emptyset$, then $|X \cap P^j| \geq 2p_j + 1$.*
2. *If $|X \cap N(P^j)| \geq 1$, then there is a vertex cover X' with $X \setminus P^j = X' \setminus P^j$ and $|X' \cap P^j| \leq 2p_j$.*

Proof. Let X be a vertex cover of G. Consider some P^j and notice that the triangles $\{a^j_s, b^j_{2s}, b^j_{2s+1}\}$, $s \in [p_j]$, are vertex-disjoint and any vertex cover has to contain at least 2 vertices in each triangle. Hence, $|X \cap P^j| \geq 2p_j$ for all $j \in [m]$.

Now, suppose that $X \cap N(P^j) = \emptyset$ for some $j \in [m]$. By assumption, the p_j edges between P^j and W are not covered by $X \cap W$ and hence X must contain $\{a^j_1, a^j_2, \ldots, a^j_{p_j}\}$. Hence, X contains all a-vertices of P^j and after removing these from P^j a path on $2p_j + 2$ vertices remains. Therefore, X has to contain at least $p_j + 1$ vertices of this path. In total, X contains at least $2p_j + 1$ vertices of P^j.

Lastly, suppose that X is a vertex cover with $|X \cap N(P^j)| \geq 1$ for some $j \in [m]$. Let $s^* \in [p_j]$ be the smallest integer such that the neighbor in W of $a^j_{s^*}$ belongs to X. We define $X'_P = \{a^j_s : s \in [p_j] \setminus \{s^*\}\} \cup \{b^j_{2s} : s \in [s^*]\} \cup \{b^j_{2s+1} : s \in [p_j] \setminus [s^* - 1]\}$ and claim that X'_P is a vertex cover of $G[P^j]$. Notice that X'_P contains 2 vertices in each triangle $\{a^j_s, b^j_{2s}, b^j_{2s+1}\}$, $s \in [p_j]$, and the edges $\{b^j_s, b^j_{s+1}\}$ are covered by the b^j with an even subscript if $s \leq 2s^*$ and otherwise by the b^j with an odd subscript. Hence, X'_P is a vertex cover of $G[P^j]$. We define $X' = (X \setminus P^j) \cup X'_P$. All edges between P^j and W are covered by $X' \cap P^j$ except the edge incident to $a^j_{s^*}$ which is covered by $X' \cap W$ by assumption. Therefore, X' is a vertex cover of G with $X \setminus P^j = X' \setminus P^j$ and $|X' \cap P^j| = |X'_P| = (p_j - 1) + s^* + (p_j + 1 - s^*) = 2p_j$. □

Lemma 12.1.3. *Let $q \in \mathbb{N}$ and $(U, \mathcal{F})$ be a q-*Hitting Set *instance with $|U| = n$ and $|\mathcal{F}| = m$. There is a hitting set for $(U, \mathcal{F})$ of size at most $\overline{h}$ if and only if $G = G(U, \mathcal{F})$ has a vertex cover of size at most $\overline{b} = \overline{h} + 2\sum_{j \in [m]} p_j$.*

Proof. Let H be a hitting set of $(U, \mathcal{F})$ of size $|H| \leq \overline{h}$. Let $X_W = \{w_i \in W : u_i \in H\}$ and consider $X = X_W \cup \bigcup_{j=1}^{m} P^j$. The set X is certainly a vertex cover of G and by applying the second part of Lemma 12.1.2 for every $j \in [m]$, which is possible since H was a hitting set, we obtain a vertex cover X' with $X' \cap W = X_W$ and $|X' \cap P^j| \leq 2p_j$ for all $j \in [m]$. Hence, X' is a vertex cover of G of size at most $\overline{h} + 2\sum_{j \in [m]} p_j$.

For the other direction, let X be a vertex cover of G of size at most $\overline{h} + 2\sum_{j\in[m]} p_j$. Due to Lemma 12.1.2, we have that $|X \cap P^j| \geq 2p_j$ for all $j \in [m]$. Define $J_X = \{j \in [m] : |X \cap P^j| > 2p_j\}$ and $X_W = X \cap W$. The size constraint implies that $|J_X| + |X_W| \leq \overline{h}$. If $J_X = \emptyset$, then by the first part of Lemma 12.1.2 we must have that $X \cap N(P^j) \neq \emptyset$ for all $j \in [m]$. By the construction of G this means that $H = \{u_i \in U : w_i \in X_W\}$ must be a hitting set for $(U, \mathcal{F})$ of size at most $\overline{h}$.

If $J_X \neq \emptyset$, then take some $j \in J_X$ and some $w_i \in N(P^j)$ and construct the vertex cover $X' = X \cup \{w_i\}$. By the second part of Lemma 12.1.2, there is a vertex cover X'' with $X'' \setminus P^j = X' \setminus P^j$ and $|X'' \cap P^j| \leq 2p_j$. This implies that $|X''| = |X| + 1 + |X'' \cap P^j| - |X \cap P^j| \leq |X| + 1 + 2p_j - (2p_j + 1) = |X|$ and $J_{X''} = J_X \setminus \{j\}$. We replace X with X'' and repeat this argument until $J_X = \emptyset$ and then obtain the desired hitting set by the previous argument. □

Lemma 12.1.4. *It holds that* $\mathrm{pw}(P^j) = \mathrm{tw}(P^j) = 2$ *for all* $j \in [m]$.

Proof. We construct a path decomposition $\mathcal{T}$ for P^j of width 2. Let $\mathcal{T}$ be a path on $2p_j + 1$ vertices $t_1, t_2, \ldots t_{2p_j+1}$. For $s \in [p_j + 1]$, define the bag $B_{t_{2s-1}} = \{b^j_{2s-1}, b^j_{2s}\}$, and for $s \in [p_j]$ define bags $B_{t_{2s}} = \{a^j_s, b^j_{2s}, b^j_{2s+1}\}$. Using these bags, $\mathcal{T}$ is a path decomposition of width 2 for P^j. Since P^j is neither a linear forest nor a forest, we have that $\mathrm{pw}(P^j) \geq 2$ and $\mathrm{tw}(P^j) \geq 2$. □

Proof of Theorem 12.1.1. Suppose that Vertex Cover can be solved in time $\mathcal{O}^*((2-\varepsilon)^{|M|})$ for some $\varepsilon > 0$, where M is a modulator to pathwidth at most 2. We argue that we can then solve q-Hitting Set in time $\mathcal{O}^*((2-\varepsilon)^n)$ for all q, thus violating SETH by Theorem 2.1.2.

Fix an arbitrary integer $q \geq 1$ and a q-Hitting Set instance $(U, \mathcal{F})$ with $|U| = n$ and $|\mathcal{F}| = m$. From $(U, \mathcal{F})$ construct the graph $G = G(U, \mathcal{F}) = (V, E)$ as described above in polynomial time. Let $M = W$, then $G - M$ is the disjoint union of the P^j, $j \in [m]$, and by Lemma 12.1.4 the pathwidth of $G - M$ is 2. So, M is a modulator to pathwidth at most 2 of size $|M| = |W| = n$. Running the Vertex Cover algorithm on G with budget $\overline{b} = \overline{h} + 2\sum_{j\in[m]} p_j$ and modulator M solves q-Hitting Set by Lemma 12.1.3. As the size of G is polynomial in $|U| = n$ and $|\mathcal{F}| = m$, the resulting running time is $\mathcal{O}^*((2-\varepsilon)^n)$ and hence SETH must be false. □

Corollary 12.1.5. *If* Vertex Cover *can be solved in time* $\mathcal{O}^*((2-\varepsilon)^{\mathrm{td}(G)})$ *for some* $\varepsilon > 0$, *then SETH is false.*

Proof. Follows from Theorem 12.1.1 and Corollary 2.4.18 □

12.2 Further Consequences

Corollary 12.2.1. *If* Maximum Cut *can be solved in time* $\mathcal{O}^*((2-\varepsilon)^{|M|})$ *for some* $\varepsilon > 0$, *where* M *is a modulator to treewidth at most 2, then SETH is false.*

Proof. We use a well-known reduction from Vertex Cover to Maximum Cut by Garey et al. [92]. Let $G = (V, E)$ be a graph. We construct another graph $G' = (V', E')$ as follows: The vertex set is given by $V' = V \cup \{x\} \cup \{e_u, e_v : e = \{u, v\} \in E\}$. The vertex x is adjacent

to all $v \in V$ and for every edge $e = \{u, v\} \in E$, we add the five edges $\{x, e_u\}$, $\{x, e_v\}$, $\{e_u, e_v\}$, $\{e_u, u\}$, and $\{e_v, v\}$ to E'. Now, G contains a vertex cover of size at most $|V(G)| - \bar{b}$ if and only if G' contains a cut of size at least $4|E(G)| + \bar{b}$.

Let M be a modulator to treewidth at most 2 for G, i.e., $\mathrm{tw}(G - M) \leq 2$. We claim that $M' = M \cup \{x\}$ is a modulator to treewidth at most 2 for G', i.e., $\mathrm{tw}(G' - M') \leq 2$. After removing M' only edges of the form $\{e_u, e_v\}$, $\{e_u, u\}$, and $\{e_v, v\}$ remain. Notice that $G' - M'$ can be obtained from $G - M$ by subdividing each edge twice, i.e., replacing each edge with a path of length 3. It is well-known that subdividing edges does not affect the treewidth, hence we have that $\mathrm{tw}(G' - M') = \mathrm{tw}(G - M) \leq 2$.

If we can solve MAXIMUM CUT in time $\mathcal{O}^*((2 - \varepsilon)^{|M|})$ for some $\varepsilon > 0$, where M is a modulator to treewidth at most 2, then we can also solve VERTEX COVER in time $\mathcal{O}^*((2 - \varepsilon)^{|M|})$ by using the discussed polynomial-time reduction and noting that any modulator to pathwidth at most 2 is also a modulator to treewidth at most 2. Hence, SETH must be false by Theorem 12.1.1. □

Corollary 12.2.2. *If* MAXIMUM CUT *can be solved in time* $\mathcal{O}^*((2 - \varepsilon)^{\mathrm{td}(G)})$ *for some* $\varepsilon > 0$*, then SETH is false.*

Proof. Follows from Corollary 12.2.1 and Corollary 2.4.18. □

Corollary 12.2.3. *Let* $r \geq 3$*. If* K_q*-*FREE DELETION *can be solved in time* $\mathcal{O}^*((2 - \varepsilon)^{|M|})$ *for some* $\varepsilon > 0$*, where* M *is a modulator to treewidth at most* $r - 1$*, then SETH is false.*

Proof. Let $G = (V, E)$ be a VERTEX COVER instance. We construct $G' = (V', E')$ by using a classical trick and replacing each edge of G with a clique on r vertices which is just like the deletion edges for the DELETION TO q-COLORABLE lower bounds. It can be seen that G has a vertex cover of size at most $\bar{b}$ if and only if there is a set $X \subseteq V'$, $|X| \leq \bar{b}$, such that $G' - X$ is K_r-free.

If M is a modulator to treewidth at most 2 for G, i.e., $\mathrm{tw}(G - M) \leq 2$, then we claim that M is also a modulator to treewidth at most $r - 1$ for G'. Let $\mathcal{T}$ be a tree decomposition of width 2 for $G - M$ with bags B_t, $t \in V(\mathcal{T})$. We construct a tree decomposition $\mathcal{T}'$ of width $r - 1$ for $G' - M$ as follows. We start with the tree decomposition $\mathcal{T}$ of $G - M$. For every edge $e \in E(G - M)$ there is a $t \in V(\mathcal{T})$ such that $e \subseteq B_t$ and to t we attach a degree-1 node t^* and the bag of t^* contains the r-clique that replaced edge e. Hence, every edge of $G' - M$ is captured in some bag of $\mathcal{T}'$ and $\mathcal{T}'$ is a tree decomposition of width $r - 1$.

Since the construction of G' can be performed in polynomial time and since the size of the modulator does not change, the result follows by Theorem 12.1.1. □

Corollary 12.2.4. *Let* $r \geq 3$*. If* K_r*-*FREE DELETION *can be solved in time* $\mathcal{O}^*((2 - \varepsilon)^{\mathrm{td}(G)})$ *for some* $\varepsilon > 0$*, then SETH is false.*

Proof. Follows from Corollary 12.2.3 and Corollary 2.4.18. □

The lower bounds obtained for VERTEX COVER, MAXIMUM CUT, and K_r-FREE DELETION are tight in the sense that we have algorithms with matching running time by straightforward dynamic programming on tree decompositions.

13 Lower Bound for Deletion to q-Colorable

In this chapter, we prove the two tight parameterized lower bounds for DELETION TO q-COLORABLE. We begin by outlining the main ideas behind our constructions in Section 13.1. Compared to the outline, we present the detailed lower bounds in reverse order, i.e., we first present the dense setting in Section 13.3 and then the sparse setting in Section 13.4. Since the constructions are very similar, this allows us to give the more complicated proof for the dense setting in full detail, while only discussing the changes needed for the sparse setting. Before presenting the lower bounds, we construct a family of $(q+1)$-critical graphs in Section 13.2 that will serve as important gadgets in both lower bound constructions.

13.1 Outline of Modulator Lower Bounds

We outline our two main results, i.e., tight lower bounds for DELETION TO q-COLORABLE parameterized by a (twinclass-)modulator to treewidth q. We begin by informally discussing the main technical obstacle to achieving such results and how we overcome it. Afterwards, we give a high-level overview of our lower bound constructions.

13.1.1 Technical Obstacle

Recall the lower bound approach relative to width parameters of Lokshtanov et al. [126] presented in Section 3.2.1. The creation of long paths in this approach is necessary to avoid the issues caused by the occurrence of undesired state transitions, which we also call *cheats*. These cheats can occur because, for example, in ODD CYCLE TRANSVERSAL it is locally always preferable to delete a vertex instead of not deleting it. In contrast, for problems such as q-COLORING, all states are equally constraining and such cheats can be avoided, hence enabling us to prove the same lower bounds under more restrictive parameters such as feedback vertex set. But for *vertex deletion problems*, like ODD CYCLE TRANSVERSAL, these cheats do occur and pose a big issue when trying to compress the path gadgets into a single separator M, since deletions in M are highly favorable. On a single separator M, such behavior means that one partial solution is *dominating* another and if we cannot control this behavior, then we lose the dominated partial solution for the purpose of encoding group assignments. Concretely, for ODD CYCLE TRANSVERSAL we obtain dominating partial solutions by deleting further vertices in the single separator M. The number of deletions is bounded from above by the *budget constraint*, but if we limit the number of deletions in M,

then we do not have $3^{|M|}$ partial solutions anymore and the construction may not be able to attain the desired base in the running time.

To resolve this issue we expand upon a technique of Cygan et al. [46] and construct an instance with a slightly large parameter value, i.e., a slightly larger single separator M. Thus, we can limit the number of deletions and are still able to encode sufficiently many group assignments. More precisely, we consider only partial solutions with the same number of deletions in M, hence only pairwise non-dominating partial solutions remain. We construct a *structure gadget* to enforce a lower bound on the number of deletions in M. A positive side effect is that the remaining gadgets can also leverage the structure of the partial solutions.

In the dense setting and especially for a higher number q of colors, this issue is amplified. Here, we consider the states of twinclasses, instead of single vertices, in a partial solution. For a twinclass, there is a hierarchy of dominating states: any state that does not delete all vertices in the twinclass is dominated by a state that deletes further vertices in the twinclass. For DELETION TO q-COLORABLE, the maximum number of states is achieved on a true twinclass of size q and we can partition the states into *levels* based on the number of deletions they induce. Within each level, the states are pairwise nondominating. Consequently, we restrict the family of partial solutions so that for every level the number of twinclasses with that level is fixed. This requires a considerably more involved construction of the structure gadget which now has to distinguish states based on their level.

13.1.2 Outline of Construction

Conceptually, the constructions for the sparse setting and for the dense setting are similar. The most significant change is in the *structure gadget*, since we have to enforce a considerably more involved structure in the dense setting. We give an overview of both settings and go into more detail for the dense case.

We fix the number of colors $q \geq 2$. *Solutions* are functions $\varphi\colon V(G) \to [q] \cup \{\bot\}$ so that for every edge $\{u, v\} \in E(G)$ either $\varphi(u) = \varphi(v) = \bot$ or $\varphi(u) \neq \varphi(v)$. Hence, $\varphi^{-1}(\bot)$ is the set of deleted vertices, whereas $\varphi\big|_{V(G)\setminus\varphi^{-1}(\bot)}$ is a q-coloring of the remaining graph.

In both settings, we want to simulate a *logical OR* constraint. For ODD CYCLE TRANSVERSAL, i.e. $q = 2$, we can use *odd cycles*. For $q \geq 3$, Theorem 13.1.1 provides an analog, where a graph H is $(q+1)$*-critical* if $\chi(H) = q+1$ and $\chi(H - v) = q$ for all $v \in V(H)$.

Theorem 13.1.1 (see Section 13.2)**.** *There exists a family $\mathcal{H}^q$ of $(q+1)$-critical graphs with treewidth q such that for every $s \in \mathbb{N}$, there exists a graph $H \in \mathcal{H}^q$ with $s \leq |V(H)| \leq s + q$.*

Setup. Given a q-SATISFIABILITY instance σ with n variables and m clauses, we start with the following standard step [126]: we partition the variables into $t = \lceil n/\beta \rceil$ groups of size β, where β only depends on the running time base that we want to rule out. Furthermore, we pick an integer p depending on β that represents the size of the groups in the graph.

13.1.3 Sparse Setting

Central Vertices and Solution Structure. We construct a graph G that has a solution φ for DELETION TO q-COLORABLE with cost $|\varphi^{-1}(\bot)| \leq \bar{b}$ if and only if σ is satisfiable. Converting from base $q+1$ to base 2 implies that G should admit a modulator M to treewidth q of size roughly $n \log_{q+1}(2)$. Like Cygan et al. [46], we make the modulator slightly larger, thus picking a larger p. The modulator M consists of $t+1$ vertex groups: the first t groups U_i, $i \in [t]$, are independent sets of size p each and correspond to the variable groups; the last group F is a clique of size q which simulates LIST COLORING constraints.

On each group U_i, we consider the set of partial solutions $\Phi_i = \{\varphi \colon U_i \to [q] \cup \{\bot\} : |\varphi^{-1}(\bot)| = p/(q+1)\}$. By picking p large enough, Φ_i is sufficiently large to encode all assignments of the i-th variable group. Defining Φ_i in this way achieves two things: first, the solutions in Φ_i are pairwise non-dominating; secondly, this fixes the budget used on the modulator. The second point is important, because by also fixing the budget on the remaining graph via a vertex-disjoint packing $\mathcal{P}$ of $(q+1)$-critical graphs, no vertex of F can be deleted, which allows us to simulate LIST COLORING constraints with the clique F.

Structure Gadgets. The next step is to enforce that only the solutions in Φ_i can be attained on group U_i. By choosing the budget $\bar{b}$ appropriately, we obtain an upper bound on the number of deletions in U_i. To obtain a lower bound, we construct the *structure gadgets*. These are built by combining $(q+1)$-critical graphs with the *arrow* gadget of Lokshtanov et al. [126]. A (thin) arrow simply propagates a deletion from a vertex u to another vertex v; else if u is not deleted, then v is not deleted and the arrow does not affect the remaining graph.

The structure gadget works as follows: if φ deletes less than $p/(q+1)$ vertices in group U_i, then there is a subset $S \subseteq U_i$ of size $|S| = (|U_i| - p/(q+1)) + 1$ that avoids all deletions in U_i. For every subset of this size, G contains a $(q+1)$-critical graph $L_{i,S}$ with an arrow from every $u \in S$ to a private vertex v in $L_{i,S}$, hence simulating an OR on the vertices in S. Since S avoids all deletions of φ, no deletion is propagated to $L_{i,S}$ and φ must pay extra to resolve $L_{i,S}$. By copying each $L_{i,S}$ sufficiently often, we can ensure that the existence of a deletion-avoiding S implies that φ must exceed our budget constraint.

Decode and Verify. The remaining construction decodes the partial solution on the modulator M and verifies if the corresponding truth assignment satisfies all clauses of σ. One could generalize the gadgets of Lokshtanov et al. [126] to higher q, but this leads to an involved construction with a worse bound on the treewidth of the remainder: for ODD CYCLE TRANSVERSAL the construction of Lokshtanov et al. has treewidth 4, whereas the simpler construction we use has only treewidth 2. More details will be presented in the dense case.

13.1.4 Dense Setting

We now have a twinclass-modulator $\mathcal{M}$ to treewidth q instead of a basic modulator and this changes the possible states as follows. Whereas φ could assume $q+1$ different states on a

single vertex u, i.e., one of the q colors or deleting the vertex, there are 2^q possible states on a true twinclass U of size q; each corresponds to a possible value of $\varphi(U) \setminus \{\perp\} \subseteq [q]$. Since U is a true twinclass, no color is used multiple times and the exact mapping $\varphi\big|_U$ is irrelevant.

Central Twinclasses and Setup. The twinclass-modulator $\mathcal{M}$ of the constructed graph G consists of $t+1$ groups and each group is a family of twinclasses. The first t groups $\mathcal{U}_i$, $i \in [t]$, correspond to the variable groups and each consists of p true twinclasses of size q that are pairwise non-adjacent. The last group contains the clique F.

Solution Structure. Our family Φ_i of considered partial solutions on group $\mathcal{U}_i$ should achieve the same two things as before. First, consider the structure of states of φ on a twinclass $U \in \mathcal{U}_i$ precisely: fix a state $C = \varphi(U) \setminus \{\perp\}$ and note that all states $C' \subsetneq C$ *dominate* C if we disregard the budget constraint, i.e., φ remains a solution if we replace C by C'. After arranging the states into *levels* according to the number ℓ of deleted vertices, there is no domination between states on the same level. This motivates the following definition.

Definition 13.1.2 (informal)**.** Given rationals $0 < c_\ell < 1$, $\ell \in [0,q]$, with $\sum_{\ell=0}^{q} c_\ell = 1$, the set Φ_i consists of solutions φ on the family of twinclasses $\mathcal{U}_i$ such that for every $\ell \in [0,q]$ there are exactly $c_\ell \cdot |\mathcal{U}_i|$ twinclasses $U \in \mathcal{U}_i$ where φ deletes exactly ℓ vertices in U.

Essentially, we are only restricting how the deletions can be distributed inside the modulator; there are no restrictions on the used colors. This again fixes the budget used on the modulator, allowing us to simulate List Coloring constraints with the clique F. By picking $c_\ell = \binom{q}{\ell} 2^{-q}$, $\ell \in [0,q]$, we ensure that Φ_i contains the solutions on $\mathcal{U}_i$ where all 2^q states appear the same number of times. This enables us to choose p small enough so that the time calculations work out and simultaneously large enough so that an injective mapping $\kappa_i \colon \{0,1\}^\beta \to \Phi_i$, mapping truth assignments of the i-th variable group to solutions in Φ_i, exists.

Thick Arrows and Structure Gadgets. To enforce the structure of Φ_i, we need a gadget to distinguish different numbers of deletions inside a twinclass. We can construct such a gadget $A_\ell(U,v)$, $\ell \in [q]$, also called *thick ℓ-arrow*. See Lemma 13.1.3 for the gadget's behavior.

Lemma 13.1.3 (informal)**.** *Let U be a set of q true twins and v be a vertex that is not adjacent to U and $\ell \in [q]$. There is a gadget $A = A_\ell(U,v)$ of treewidth q with the following properties:*

- *Any solution φ must delete at least ℓ vertices in $A - U$.*
- *If a solution φ deletes exactly ℓ vertices in $A - U$, then φ can only delete v if φ deletes at least ℓ vertices in U.*

We proceed by constructing the *structure gadgets* which enforce that the partial solution on $\mathcal{U}_i$ belongs to Φ_i. Let $c_{<\ell} = c_0 + \cdots + c_{\ell-1}$ for all $\ell \in [0,q]$. For every group $i \in [t]$, number of deletions $\ell \in [q]$, set of twinclasses $\mathcal{S} \subseteq \mathcal{U}_i$ with $|\mathcal{S}| = c_{<\ell} \cdot p + 1$, we add an $(q+1)$-critical graph $L_{i,\ell,\mathcal{S}} \in \mathcal{H}^q$ consisting of at least $|\mathcal{S}|$ vertices. For every $U \in \mathcal{S}$, we pick a private vertex v in $L_{i,\ell,\mathcal{S}}$ and add the thick ℓ-arrow $A_\ell(U,v)$. We create a large number of

copies of each $L_{i,\ell,\mathcal{S}}$ and the incident thick arrows. This concludes the construction of the structure gadget.

The number of deletions in the central vertices is already bounded from above by the budget constraint. If too few deletions occur in the twinclasses of $\mathcal{U}_i$, then we can find an ℓ and an $\mathcal{S} \subseteq \mathcal{U}_i$ with $|\mathcal{S}| = c_{<\ell} \cdot p + 1$ such that less than ℓ vertices are deleted in each $U \in \mathcal{S}$. Hence, all thick ℓ-arrows leading to $L_{i,\ell,\mathcal{S}}$ and its copies cannot propagate deletions. To resolve all these $(q+1)$-critical graphs, one extra vertex per copy must be deleted. Due to the large number of copies, this implies that we must violate our budget constraint.

Hence, for any $\mathcal{S} \subseteq \mathcal{U}_i$ with $|\mathcal{S}| = c_{<\ell} \cdot p + 1$ and any solution φ obeying the budget constraint there is at least one twinclass $U \in \mathcal{S}$ in which φ deletes at least ℓ vertices. Therefore, there are at least $(1 - c_{<\ell})p$ twinclasses in $\mathcal{U}_i$ where φ deletes at least ℓ vertices. Since this holds for all $\ell \in [0, q]$ and the budget $\bar{b}$ is chosen appropriately, all inequalities have to be tight and the deletions inside $\mathcal{U}_i$ follow the distribution imposed by Φ_i. The upcoming decoding gadgets will make use of this structure.

Color-Set-Gadgets and Decoding Gadgets. Next, we discuss the decoding part of the construction. Since gadgets cannot read the color of single vertices but only of a whole twinclass, we need *color-set-gadgets* to detect the colors used on a twinclass, cf. Lemma 13.1.4.

Lemma 13.1.4 (informal)**.** *Let U be a set consisting of q true twins and v be a vertex that is not adjacent to U and let $C \subsetneq [q]$. There is a gadget $S = S_C(U, v)$ of treewidth q such that:*

- *Any solution φ deletes at least $(q - |C|) + 1$ vertices in $S - U$.*
- *If φ deletes exactly $(q - |C|) + 1$ vertices in $S - U$, then $\varphi(v) = \bot$ only if $\varphi(U) \setminus \{\bot\} \subseteq C$.*

To construct the color-set-gadgets we rely on the List Coloring constraints that are simulated with the central clique F. Note that the color-set-gadgets only check for set inclusion and not set equality. Using the structure of solutions in Φ_i however, the color-set-gadgets will still be sufficient to distinguish the solutions in Φ_i from each other.

By using a complete $(q+1)$-partite graph with all sets of the partition being singletons except for one large independent set, we can simulate a logical AND, see Lemma 13.1.5.

Lemma 13.1.5 (informal)**.** *Let n_Y be a positive integer. There is a gadget Y of treewidth q with a set of input vertices $V' \subseteq V(Y)$, $|V'| = n_Y$, and a vertex $\hat{y} \in V(Y) \setminus V'$ such that:*

- *Any solution φ has to delete at least one vertex in $Y - V'$.*
- *If φ deletes exactly one vertex in $Y - V'$, then $\varphi(\hat{y}) = \bot$ only if $\varphi(V') = \{\bot\}$.*

For the j-th clause, variable group $i \in [t]$, solution $\varphi_i \in \Phi_i$, we invoke Lemma 13.1.5 to create a gadget Y^j_{i,φ_i} for $n_Y = (1 - c_r)p = (1 - 2^{-q})p$ input vertices and with distinguished vertex $\hat{y}^j_{i,\varphi_i}$. For every twinclass $U \in \mathcal{U}_i$ with $\varphi_i(U) \neq [q]$, we pick a private input vertex v of Y^j_{i,φ_i} and add the color-set-gadget $B_{\varphi_i(U)\setminus\{\bot\}}(U, v)$. By Lemma 13.1.5, the vertex $\hat{y}^j_{i,\varphi_i}$ can only be deleted if all input vertices of Y^j_{i,φ_i} are deleted. Due to Lemma 13.1.4 and the structure of Φ_i, this will only be the case if φ_i is the partial solution on $\mathcal{U}_i$.

Clause Gadgets. For the j-th clause, we add an $(q+1)$-critical graph $Z^j \in \mathcal{H}^q$ consisting of at least $q2^\beta$ vertices. For every group $i \in [t]$ and solution $\varphi_i \in \Phi_i$ such that $\kappa_i^{-1}(\varphi_i)$ is a partial truth assignment satisfying the j-th clause, we pick a private vertex v in Z^j and add

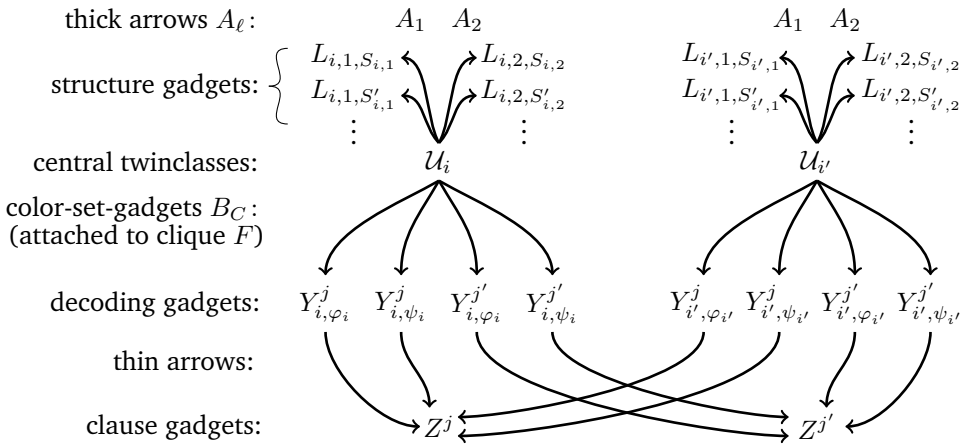


Fig. 13.1.: An overview of the construction for the dense setting in case of $q = 2$. The arrows point in the direction in that deletions are propagated by the corresponding gadget.

a thin arrow from $\hat{y}^j_{i,\varphi_i}$ to v. The budget constraint will ensure that the only way to delete a vertex in Z^j is by propagating a deletion via a thin arrow from some $\hat{y}^j_{i,\varphi_i}$. By construction of the decoding and clause gadgets, this is only possible if the partial solution on $\mathcal{U}_i$ corresponds to a satisfying assignment of the j-th clause. This concludes the construction, cf. Figure 13.1.

Budget and Packing. The budget $\overline{b} = \overline{b}_0 + \mathrm{cost}_{\mathcal{P}}$ of the constructed instance $(G, \overline{b})$ consists of two parts; $\overline{b}_0 = tqp/2$ is allocated to the central twinclasses and matches the number of deletions incurred by picking a partial solution $\varphi_i \in \Phi_i$ on $\mathcal{U}_i$ for each group $i \in [t]$; the second part $\mathrm{cost}_{\mathcal{P}}$ is due to a vertex-disjoint packing $\mathcal{P}$ which we describe next. A part of each thin arrow in G is added to $\mathcal{P}$ and for every thick arrow, color-set-gadget, or decoding gadget, we add the appropriate parts to $\mathcal{P}$ given by Lemmas 3.3, 3.4, 3.5, respectively. Summing up the implied costs yields $\mathrm{cost}_{\mathcal{P}}$. Hence, we know how the deletions are distributed throughout the various gadgets. In particular, this ensures that no vertex of the central clique F is deleted.

Theorem 10.2.5 follows by using these ideas and working out the remaining technical details.

13.2 Construction of Critical Graphs

We say that a graph G is *t-critical* if $\chi(G) = t$ and $\chi(G - v) = t - 1$ for all $v \in V(G)$. The odd cycles form a family of 3-critical graphs of treewidth 2 that contains graphs of arbitrarily large size. To obtain lower bounds for DELETION TO q-COLORABLE with $q > 2$ instead of ODD CYCLE TRANSVERSAL, we instead construct a family of $(q+1)$-critical graphs of treewidth q that contains graphs of arbitrarily large size.

Theorem 13.2.1. *Let $q \geq 2$. There exists a family $\mathcal{H}^q$ of graphs such that*

- *H is $(q+1)$-critical for all $H \in \mathcal{H}^q$,*
- *$\mathrm{pw}(H) = \mathrm{tw}(H) = q$ for all $H \in \mathcal{H}^q$,*
- *for every $s \in \mathbb{N}$, there exists a graph $H \in \mathcal{H}^q$ with $s \leq |V(H)| \leq s + q$.*

Definition 13.2.2 ([55, 98]). Given two graphs G and H and edges $\{v,w\} \in E(G)$ and $\{x,y\} \in E(H)$. We obtain a graph $G' = (V', E')$ by applying *Hajós' construction*, where $V' = (V(G) \cup V(H) \cup \{s\}) \setminus \{v,x\}$ and $E' = E(G-v) \cup E(H-x) \cup \{\{s,u\} : \{v,w\} \neq \{v,u\} \in E(G)\} \cup \{\{s,u\} : \{x,y\} \neq \{x,u\} \in E(H)\} \cup \{w,y\}$. That is, we remove the edges $\{v,w\}$ and $\{x,y\}$, identify v and x into a single vertex called s, and add the edge $\{w,y\}$.

Definition 13.2.3 ([55, 98]). Let $t \geq 3$. A graph $G = (V,E)$ is *t-constructible* if it can be constructed from the following operations, starting with the complete graph K_t:

- Let G and H be t-constructible graphs, then the graph obtained by applying Hajós' construction to G and H (wrt. edges $\{v,w\} \in E(G)$ and $\{x,y\} \in E(H)$ and identifying v and x) is t-constructible.
- Let G be a t-constructible graph and u and v two non-adjacent vertices in G. The graph formed by adding the edge $\{u,v\}$ to G and then contracting $\{u,v\}$ is t-constructible.

Lemma 13.2.4 ([55, 98]). *If a graph G is t-constructible, then $\chi(G) \geq t$, i.e., coloring G requires at least t colors.*

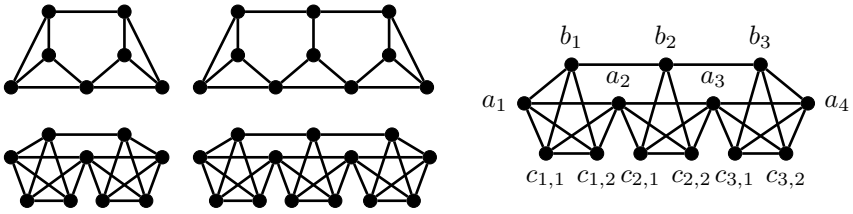


Fig. 13.2.: The graphs $H_2^4, H_3^4, H_2^5, H_3^5$ and the vertex labels of H_3^5.

We set $H_1^t = K_t$ with vertex labels $V(H_1^t) = \{a_1, a_1', b_1, c_{1,1}, c_{1,2}, \ldots, c_{1,t-3}\}$. For $\gamma \geq 2$, we obtain H_γ^t by performing Hajós' construction on $H_{\gamma-1}^t$ and K_t, where we label the vertices of K_t by $a_\gamma, a_\gamma', b_\gamma, c_{\gamma,1}, c_{\gamma,2}, \ldots, c_{\gamma,t-3}$, with respect to the edges $\{b_{\gamma-1}, a_{\gamma-1}'\} \in E(H_{\gamma-1}^t)$ and $\{a_\gamma, b_\gamma\} \in E(K_t)$ and identifying $a_{\gamma-1}'$ with a_γ. Having finished the construction, we will not require the a'-labels anymore, in H_γ^t we set $a_{\gamma+1} = a_\gamma'$, see Figure 13.2.

We define $\mathcal{H}^{t-1} = \{H_\gamma^t : \gamma \in \mathbb{N}\}$ for $t \geq 3$. Observe that for $t = 3$ we obtain exactly the odd cycles. We establish the following properties of the graphs H_γ^t which directly implies Theorem 13.2.1.

Lemma 13.2.5. *The graphs H_γ^t, $t \geq 3$, $\gamma \in \mathbb{N}$, have the following properties:*

1. $|V(H_\gamma^t)| = (t-1)\gamma + 1$. ***2.*** *$H_\gamma^t$ is t-critical.* ***3.*** $\mathrm{pw}(H_\gamma^t) = \mathrm{tw}(H_\gamma^t) = t-1$.

Proof. Property 1 can be easily shown by induction over γ. Now, we can prove that H_γ^t is t-critical. By Lemma 13.2.4 we have that $\chi(H_\gamma^t) \geq t$. Let $v \in V(H_\gamma^t)$ be arbitrary. We have to show that $\chi(H_\gamma^t - v) \leq t-1$. We distinguish whether v is labeled by a, b, or c, see Figure 13.3.

Case $v = a_i$, $i \in [\gamma+1]$**:** We construct a proper $(t-1)$-coloring $\varphi\colon V(H_\gamma^t - v) \to [t-1]$. For $\ell \in [\gamma]$, set $\varphi(b_\ell) = (\ell \mod 2) + 1$. For $\ell \in [\gamma+1]$, set $\varphi(a_\ell) = (\ell+1 \mod 2) + 1$ if $\ell < i$ and set $\varphi(a_\ell) = (\ell \mod 2) + 1$ if $\ell > i$. For $\ell \in [\gamma]$ and $k \in [t-3]$, set $\varphi(c_{\ell,k}) = k+2$. Now, for all $\ell \in [\gamma]$, the $(t-1)$-clique induced by $a_\ell, a_{\ell+1}, c_{\ell,1}, \ldots, c_{\ell,t-3}$ is colored properly and

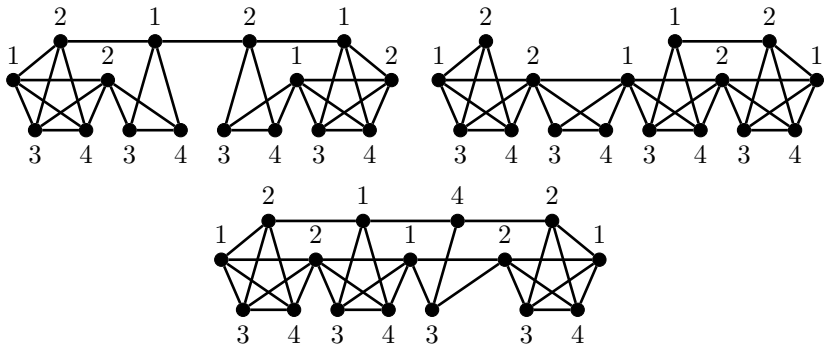


Fig. 13.3.: The three coloring cases in the proof of Lemma 13.2.5 for the graph H_4^5.

the graph induced by $N[b_\ell]$ is colored properly. Since these cover all edges, φ must be a proper coloring.

Case $v = b_i$, $i \in [\gamma]$: Again, we construct a proper $(t-1)$-coloring φ. For $\ell \in [\gamma]$, set $\varphi(b_\ell) = (\ell \mod 2) + 1$ if $\ell < i$ and set $\varphi(b_\ell) = (\ell + 1 \mod 2) + 1$ if $\ell > i$. For $\ell \in [\gamma + 1]$, set $\varphi(a_\ell) = (\ell + 1 \mod 2) + 1$. For $\ell \in [\gamma]$ and $k \in [t-3]$, set $\varphi(c_{\ell,k}) = k + 2$. Similarly to the previous case, φ must be a proper coloring.

Case $v = c_{i,k}$, $i \in [\gamma]$, $k \in [t-3]$: We construct a proper $(t-1)$-coloring φ as follows. We set $\varphi(b_i) = k + 2$. For $\ell \in [\gamma] \setminus \{i\}$, set $\varphi(b_\ell) = (\ell \mod 2) + 1$ if $\ell < i$ and set $\varphi(b_\ell) = (\ell + 1 \mod 2) + 1$ if $\ell > i$. For $\ell \in [\gamma + 1]$, set $\varphi(a_\ell) = (\ell + 1 \mod 2) + 1$. For $(\ell, k') \in ([\gamma] \times [t-3]) \setminus \{(i,k)\}$, set $\varphi(c_{\ell,k'}) = k' + 2$. Again, like in the previous cases, φ must be a proper coloring.

This shows that H_γ^t is t-critical. We proceed by showing that $\mathrm{pw}(H_\gamma^t) = \mathrm{tw}(H_\gamma^t) = t - 1$. Note that by contracting all vertices in $V(H_\gamma^t) \setminus \{a_1, a_2, b_1, c_{1,1}, \ldots, c_{1,t-3}\}$ into b_1, we obtain K_t and hence $\mathrm{pw}(G) \geq \mathrm{tw}(G) \geq \mathrm{tw}(K_t) = t - 1$, as treewidth is a minor-monotone parameter. For the other direction, we construct a path decomposition $\mathcal{T}$ of width $t-1$. Let $\mathcal{T}$ be the path on $2\gamma - 1$ vertices denoted by $w_1, \ldots, w_{2\gamma-1}$. For $\ell \in [\gamma]$, we define the bag $B_{w_{2\ell-1}} = \{a_\ell, a_{\ell+1}, b_\ell, c_{\ell,1}, c_{\ell,2}, \ldots, c_{\ell,t-3}\}$ and for $\ell \in [\gamma - 1]$, we define the bag $B_{w_{2\ell}} = \{b_\ell, b_{\ell+1}, a_{\ell+1}\}$. This yields a path decomposition of width $t-1$ and hence establishes property 3. □

13.3 Dense Setting

We again view *solutions* to Deletion to q-Colorable as functions $\varphi\colon V(G) \to [q] \cup \{\bot\}$ satisfying $\varphi(u) = \bot$ or $\varphi(v) = \bot$ or $\varphi(u) \neq \varphi(v)$ for every edge $\{u, v\} \in E(G)$.

We will prove a lower bound for Deletion to q-Colorable parameterized by the size of a twinclass-modulator (TCM) to treewidth q. This implies a lower bound for parameterization by twinclass-treedepth due to Lemma 2.4.23. By Section 2.4.4, this further extends to lower bounds for parameterization by twinclass-pathwidth and clique-width. Hence, we will see that the running time of the algorithm from Theorem 14.0.1 is tight, unless CNF-SETH is false.

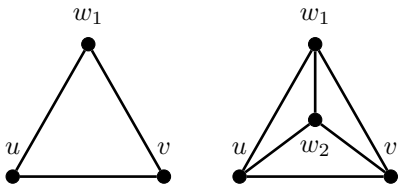


Fig. 13.4.: A deletion edge between u and v for $q = 2$ and $q = 3$.

Theorem 13.3.1. *If* DELETION TO q-COLORABLE *can be solved in time* $\mathcal{O}^*((2^q - \varepsilon)^{|\mathcal{M}|})$ *for some* $q \geq 2$ *and* $\varepsilon > 0$*, where* $\mathcal{M}$ *is a TCM to treewidth* q*, then CNF-SETH is false.*

Assume that we can solve DELETION TO q-COLORABLE in time $\mathcal{O}^*((2^q - \varepsilon)^{|\mathcal{M}|})$ for some $q \geq 2$ and $\varepsilon > 0$. We provide a reduction from SATISFIABILITY with n variables and m clauses to DELETION TO q-COLORABLE with a TCM to treewidth q of size approximately $n \log_{2^q}(2) = n/q$ to convert from base 2^q to base 2 and graph size polynomial in n and m. Combining this reduction with the assumed algorithm will imply a faster algorithm for SATISFIABILITY, thus violating CNF-SETH. From now on, we consider q to be fixed.

Before we begin with the actual construction, we describe several gadgets that will be important.

13.3.1 Preliminary Gadgets

Deletion Edges. Let u and v be two vertices. By adding a *deletion edge* between u and v, we mean adding $q - 1$ vertices $w_1, \ldots, w_{q-1}$ and edges so that $\{u, v, w_1, w_2, \ldots, w_{q-1}\}$ is a clique of size $q + 1$, see Figure 13.4. The vertices $w_1, \ldots, w_{q-1}$ will not receive any further incident edges. Therefore, any solution to DELETION TO q-COLORABLE has to delete at least one vertex in this clique and we can assume that the deleted vertex is u or v.

Arrows. Our construction relies on being able to propagate deletions throughout the graph. This is done via so-called *arrows*. We distinguish between two types of arrows, namely *thin arrows* and *thick arrows*. Thin arrows arise already in the sparse setting of the lower bound construction of Lokshtanov et al. [126] for the restricted case of $q = 2$, i.e., ODD CYCLE TRANSVERSAL[pathwidth]. A thin arrow simply propagates a deletion from a single vertex to another vertex. Whereas the newly introduced thick arrows propagate a deletion to a single vertex only if sufficiently many vertices have been deleted in some true twinclass of size q. The construction of thick arrows is described in the proof of Lemma 13.3.2.

Let us proceed with the construction of thin arrows. Given two single vertices u and v, *adding a thin arrow* from u to v means adding a new vertex w and a deletion edge between u and w and a deletion edge between w and v. We remark that compared to the construction of Lokshtanov et al. [126], we have shortened the thin arrows. The construction of thin arrows is symmetric, but the direction will be important later for the description of a packing that witnesses a lower bound on the required budget. The idea is that when u is not deleted, we delete w to resolve both incident deletion edges; when u is deleted, the first deletion edge is resolved and we can afford to delete v to resolve the second deletion edge. The former is called the *passive* solution of the thin arrow and the latter is the *active* solution.

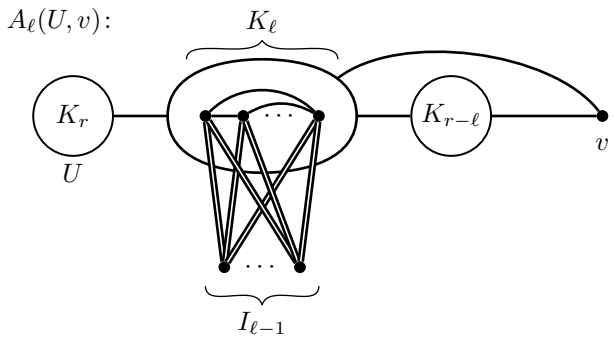


Fig. 13.5.: The construction of a thick ℓ-arrow $A_\ell(U,v)$. The double-lined edges represent deletion edges. An edge attached to a circle or ellipse represents a *join*, i.e., a bundle of edges.

Using exchange arguments, one can see that it is sufficient to only consider solutions that on each thin arrow use either the passive solution or active solution.

Lemma 13.3.2. *Let U be a set of q true twins and v be a vertex that is not adjacent to U and $\ell \in [q]$. There is a graph $A = A_\ell(U,v)$ with the following properties:*

- *Any solution φ satisfies $|V(A-U) \cap \varphi^{-1}(\perp)| \geq \ell$.*
- *If $|V(A-U) \cap \varphi^{-1}(\perp)| = \ell$, then $\varphi(v) = \perp$ implies that $|U \cap \varphi^{-1}(\perp)| \geq \ell$.*
- *Any solution $\tilde{\varphi}$ for $\tilde{G} = G[(V \setminus V(A)) \cup U \cup \{v\}]$ with $\tilde{\varphi}(v) \neq \perp$ or $|U \cap \tilde{\varphi}^{-1}(\perp)| \geq \ell$ can be extended to a solution φ for G with $|V(A-U) \cap \varphi^{-1}(\perp)| = \ell$.*
- *$\mathrm{tw}(A-U) \leq q$ via a winning cops-and-robber-strategy starting with a cop on v.*

Proof. The graph $A = A_\ell(U,v)$ is constructed by adding a clique K_ℓ on ℓ vertices, a clique $K_{q-\ell}$ on $q-\ell$ vertices, an independent set $I_{\ell-1}$ on $\ell-1$ vertices and adding all edges between U and K_ℓ, all edges between K_ℓ and $K_{q-\ell}$, all edges between K_ℓ and v, all edges between $K_{q-\ell}$ and v, and a deletion edge between each vertex of K_ℓ and each vertex of $I_{\ell-1}$, cf. Figure 13.5. Observe that the K_ℓ, $K_{q-\ell}$, and v together form a clique of size $q+1$. Only U and v can have further incident edges.

We first show that any solution φ has to delete at least ℓ vertices in $A-U$. Suppose that no deletion occurs in K_ℓ, then $\ell-1$ disjoint deletion edges between K_ℓ and $I_{\ell-1}$ remain and at least one deletion has to occur in $K_{q-\ell} \cup \{v\}$. On the other hand, suppose that $1 \leq d \leq \ell$ deletions occur in K_ℓ, then $\ell - d$ disjoint deletion edges between K_ℓ and $I_{\ell-1}$ remain. In either case, we must delete at least ℓ vertices.

Next, we argue that a solution φ deleting exactly ℓ vertices in $A-U$ cannot delete v if less than ℓ deletions occur in U. Suppose that $d < \ell$ deletions occur in U, then at least $\ell - d \geq 1$ deletions must occur in K_ℓ, because U and K_ℓ together form a clique of size $q+\ell$. Let $d' \geq 1$ be the number of deletions in K_ℓ, then like in the previous paragraph $\ell - d'$ disjoint deletion edges between K_ℓ and $I_{\ell-1}$ remain. Hence, we must already perform a total of ℓ deletions in K_ℓ and the deletion edges, so v cannot be deleted.

Consider any solution $\tilde{\varphi}$ for $\tilde{G}$. If $\tilde{\varphi}(v) \neq \perp$, then we can extend $\tilde{\varphi}$ to a solution φ for G by deleting K_ℓ completely. This deletes all edges between U and A and one endpoint of each deletion edge. The $K_{q-\ell}$ is only attached to v and can always be properly colored. We give all vertices in $I_{\ell-1}$ the same color and the cliques attached to $I_{\ell-1}$ due to the deletion

edges are all disjoint and of size $q-1$ each, so they can be properly colored as well. Hence, φ is a valid solution for G.

If $|U \cap \tilde{\varphi}^{-1}(\bot)| \geq \ell$ and $\tilde{\varphi}(v) = \bot$, then we can extend $\tilde{\varphi}$ to a solution φ for G by deleting $I_{\ell-1}$ completely. To K_ℓ we assign a subset of the colors $[q] \setminus \tilde{\varphi}(U)$, this is possible since at least ℓ deletions occur in U. The K_ℓ and $K_{q-\ell}$ together form a clique of size q that we can properly color. The cliques of the deletion edges are all disjoint as before and only attached to K_ℓ and have size $q-1$ each, hence they can also be properly colored. Hence, φ is a valid solution for G.

Finally, we argue that $\mathrm{tw}(A-U) \leq q$ using the omniscient cops-and-robber-game. We begin by placing $q+1$ cops on $K_\ell \cup K_{q-\ell} \cup \{v\}$, then we remove the $q-\ell+1$ cops on $K_{q-\ell} \cup \{v\}$. Now, all remaining connected components correspond to a vertex of $I_{\ell-1}$. Fix the component the robber escaped to and place one cop on the corresponding vertex of $I_{\ell-1}$. Now, the robber must have escaped to the inside of some deletion edge between K_ℓ and $I_{\ell-1}$. We remove the $\ell-1$ cops from K_ℓ that are not on the endpoint of this deletion edge. Finally, we place $q-1$ cops on the remaining vertices of the deletion edge and capture the robber. Throughout the strategy, we have never placed more than $q+1$ cops simultaneously and hence the claimed treewidth bound holds by Theorem 2.4.30. □

Similarly to the thin arrows, we say that a thick ℓ-arrow $A_\ell(U,v)$ is *active* if the considered solution φ satisfies $|U \cap \varphi^{-1}(\bot)| \geq \ell$ and hence Lemma 13.3.2 allows us to assume that $\varphi(v) = \bot$; otherwise, we say that $A_\ell(U,v)$ is *passive*.

13.3.2 Complete Construction

Construction Setup. We will now begin with the construction of the DELETION TO q-COLORABLE instance. Consider a SATISFIABILITY instance σ with n variables and m clauses. We enumerate the clauses and simply refer to them by their number. Depending only on ε and q, we will choose an integer β; we will describe how to choose β later. We partition the variables of σ into groups of size at most β, resulting in $t = \lceil n/\beta \rceil$ groups which will be indexed by i. Next, we choose the smallest integer p such that p is divisible by 2^q and $(2^q)^p \frac{(2^q-1)!}{2^{(2^q)}} p^{-(2^q)} \geq 2^\beta$.

Usually, a simple base conversion from a base that is a power of 2 to base 2 allows for a construction where the handling of groups is a lot less technical. Unfortunately, since we require the trick of Cygan et al. [46] that enlarges the groups, this is not possible here. We will now describe the construction of the DELETION TO q-COLORABLE instance $G = G(\sigma, q, \beta)$.

Central Twinclasses. The central vertices of G will form the TCM to treewidth q. For each variable group $i \in [t]$, we create a set $\mathcal{U}_i \subseteq \Pi_{tc}(G)$ consisting of p true twinclasses of size q each; there are no edges between the different twinclasses. The partial solution induced by a solution φ of DELETION TO q-COLORABLE on the twinclasses in $\mathcal{U}_i$ will correspond to a truth assignment for the i-th variable group.

Central Clique. We create a clique $F = \{f_1, \ldots, f_q\}$ on q vertices which also belongs to the central part. The vertices of F will not be twins; each vertex of F has its own twinclass. This concludes the construction of the central part which will form the modulator $\mathcal{M}$.

Our construction will ensure that no vertex of F can be deleted; we will prove so later. This allows us to use F to simulate LIST COLORING constraints, i.e., we can forbid specific colors at a vertex. To do so, we normalize the considered solutions by assuming that $\varphi(f_s) = s$ for all $s \in [q]$ and then no vertex v adjacent to some f_s can receive color s. We say that a solution φ is *normalized* if $F \cap \varphi^{-1}(\bot) = \emptyset$ and $\varphi(f_s) = s$ for all $s \in [q]$.

We mention here that in principle a twinclass-modulator could contain arbitrarily large twinclasses. In our case, every twinclass has size at most $q = \mathcal{O}(1)$, so the number $|\bigcup(\mathcal{M})|$ of vertices in $\mathcal{M}$ is linear in the size $|\mathcal{M}|$ of $\mathcal{M}$.

Budget. The budget $\bar{b} = \mathrm{cost}_{\mathcal{P}} + tqp/2$ for the DELETION TO q-COLORABLE instance is split into two parts. The first part $\mathrm{cost}_{\mathcal{P}}$ is due to a vertex-disjoint packing $\mathcal{P}$ of subgraphs that will be described later and the second part $tqp/2$ is allocated to the vertices in the twinclass-modulator.

Partial Solution Structure. We want to ensure that under our budget restriction no vertex of F can be deleted, as otherwise the simulation of LIST COLORING constraints will not work. This entails that all considered solutions should perform the same number of deletions. On the gadgets that will be attached to the central vertices, this does not pose a big issue. However, it does on the central vertices: to obtain the desired lower bound, for each true twinclass $U \in \mathcal{U}$ and solution φ, it should in principle be possible for $\varphi(U)$ to attain every subset of $[q]$, so that we have 2^q states per twinclass. However, the number of deletions in U depends on $|\varphi(U)|$. We can partition the possible states into different *levels* depending on the number ℓ of deletions they incur in the true twinclass. A state can incur any number of deletions between 0 and q.

Using slightly larger groups like Cygan et al. [46] allows us to overcome these issues. By using slightly more twinclasses per variable group $i \in [t]$ than necessitated by the base conversion, we can consider solutions with a special structure on the central vertices. It is designed in such a way that all solutions obeying this structure incur the same number of deletions on the central vertices while still allowing all states to appear the same number of times. Due to the slight increase in the number of twinclasses, we can still encode sufficiently many truth assignments into such structured solutions.

For every $i \in [t]$, $U \in \mathcal{U}_i$, $C \subseteq [q]$, we fix some representative solution $\varphi_{i,U,C} \colon U \to [q] \cup \{\bot\}$ to DELETION TO q-COLORABLE on U with $\varphi_{i,U,C}(U) \setminus \{\bot\} = C$. The point of this is that we only distinguish between different solutions φ based on the sets $\varphi(U)$, $U \in \mathcal{U}_i$, and not based on the actual mappings $\varphi\big|_U$, $U \in \mathcal{U}_i$. Since there are exactly $\binom{q}{q-\ell} = \binom{q}{\ell}$ states that incur exactly ℓ deletions each and we want to allow for the possibility of all states appearing the same number of times, we arrive at the following definition for each group $i \in [t]$. The family Φ_i consists of all solutions $\varphi \colon \bigcup(\mathcal{U}_i) \to [q] \cup \{\bot\}$ to DELETION TO q-COLORABLE on $\bigcup(\mathcal{U}_i)$ with the properties

- $\left|\left\{U \in \mathcal{U}_i : \varphi(U) \setminus \{\bot\} \in \binom{[q]}{\ell}\right\}\right| = \binom{q}{\ell}\frac{p}{2^q}$ for all $\ell \in [q]$, and
- $\varphi\big|_U = \varphi_{i,U,\varphi(U)\setminus\{\bot\}}$ for all $U \in \mathcal{U}_i$.

Informally, this means that we only consider partial solutions that pick a representative solution on each twinclass in $\mathcal{U}_i$ and for every level $\ell \in [0, q]$, i.e., a possible number of deletions, there is a fixed number $\binom{q}{\ell}\frac{p}{2^q}$ of twinclasses in $\mathcal{U}_i$ where the partial solution attains a state of level ℓ. Lemma 13.3.3 gives the total number of deletions for a partial solution $\varphi_i \in \Phi_i$ and Lemma 13.3.4 studies the size of Φ_i.

Lemma 13.3.3. *If $\varphi \in \Phi_i$, then $|\varphi^{-1}(\bot)| = qp/2$.*

Proof.

$$|\varphi^{-1}(\bot)| = \sum_{\ell=0}^{q}(q-\ell)\binom{r}{\ell}\frac{p}{2^q} = \frac{p}{2^q}q\sum_{\ell=0}^{q-1}\binom{q-1}{\ell} = \frac{p}{2^q}q2^{q-1} = qp/2. \qquad \square$$

Lemma 13.3.4. *If $p \geq 2^q$, then we have that $|\Phi_i| \geq (2^q)^p \frac{(2^q-1)!}{2^{(2^q)}} p^{-(2^q)}$.*

Proof. Observe that

$$\left|\left\{\varphi \in \Phi_i : |\{U \in \mathcal{U}_i : \varphi(U) \setminus \{\bot\} = C\}| = \frac{p}{2^q} \text{ for all } C \subseteq [q]\right\}\right| = \binom{p}{\frac{p}{2^q}, \ldots, \frac{p}{2^q}} = x,$$

where x is the central multinomial coefficient. The central multinomial coefficient x is a maximum of the function $(c_1, \ldots, c_{2^q}) \mapsto \binom{p}{c_1,\ldots,c_{2^q}}$. Hence, x is a maximum term in the sum of the multinomial theorem, i.e., $(2^q)^p = \sum_{c_1+\cdots+c_{2^q}=p}\binom{p}{c_1,\ldots,c_q}$. The number of terms in the sum of the multinomial theorem is the number of weak compositions of p into 2^q parts which is $\binom{p+2^q-1}{p}$. Bounding this term from above using $p \geq 2^q$, we obtain

$$\binom{p+2^q-1}{p} = \frac{1}{(2^q-1)!}(p+1)\cdots(p+2^q-1) \leq \frac{1}{(2^q-1)!}(2p)^{(2^q)} = \frac{2^{2^q}}{(2^q-1)!}p^{(2^q)}.$$

By the multinomial theorem, we hence obtain that

$$|\Phi_i| \geq x \geq (2^q)^p \frac{(2^q-1)!}{2^{(2^q)}} p^{-(2^q)}. \qquad \square$$

Hence, if we choose p as discussed previously, then we can pick for each group $i \in [t]$ an injective mapping $\kappa_i \colon \{0,1\}^\beta \to \Phi_i$ mapping truth assignments of the i-th variable group to partial solutions on $\mathcal{U}_i$ with the desired structure.

Structure Gadgets. We proceed by constructing the *structure gadgets* to enforce that the partial solution on $\mathcal{U}_i$ belongs to Φ_i. For every group $i \in [t]$, number of deletions $\ell \in [q]$, set of twinclasses $\mathcal{S} \subseteq \mathcal{U}_i$ with $|\mathcal{S}| = \binom{q}{\geq(q-\ell+1)}\frac{p}{2^q} + 1$, we add an $(q+1)$-critical graph $L_{i,\ell,\mathcal{S}} \in \mathcal{H}^q$ consisting of at least $|\mathcal{S}|$ vertices. For every $U \in \mathcal{S}$, we pick a private vertex v in $L_{i,\ell,\mathcal{S}}$ and add the thick ℓ-arrow $A_\ell(U, v)$. We create $1 + tqp/2 = 1 + (\overline{b} - \mathrm{cost}_{\mathcal{P}})$ copies of $L_{i,\ell,\mathcal{S}}$ and the incident thick arrows. This concludes the construction of the structure gadget, cf. Figure 13.6.

Due to our budget constraint, no solution will be able to perform too many deletions in the central vertices. The idea of the structure gadget is that if a solution φ performs too few deletions in the twinclasses of $\mathcal{U}_i$, then we can find an ℓ and an $\mathcal{S} \subseteq \mathcal{U}_i$ with

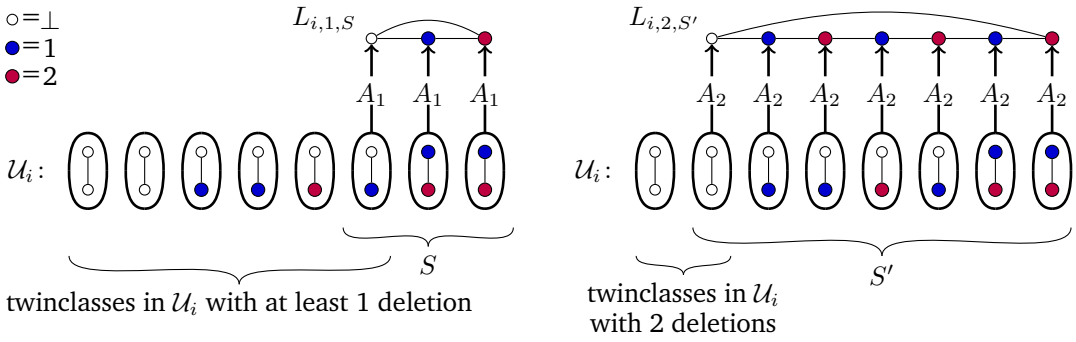


Fig. 13.6.: The construction of structure gadgets $L_{i,1,S}$ and $L_{i,2,S'}$ for the case of $q = 2$, i.e., ODD CYCLE TRANSVERSAL, and $p = 8$ and its connection to the central twinclasses. The large ellipses represent twinclasses and the arrows represent thick 1-arrows or thick 2-arrows. The depicted solution φ_i belongs to Φ_i and propagates a deletion to every attached structure gadget. Note that φ_i does not use each color set the same number of times.

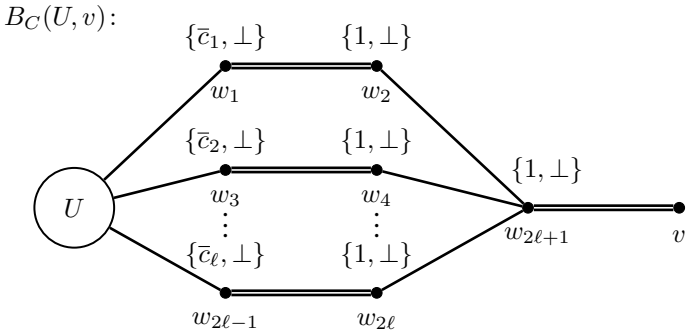


Fig. 13.7.: The construction of a color-set-gadget $S_C(U,v)$ with $[q] \setminus C = \{\overline{c}_1,\ldots,\overline{c}_\ell\}$. The double-lined edges represent deletion edges. An edge attached to a circle or ellipse represents a *join*, i.e., a bundle of edges. The sets depict the allowed states at a vertex.

$|\mathcal{S}| = \binom{q}{\geq(q-\ell+1)}\frac{p}{2^q}+1$ such that φ deletes less than ℓ vertices in each $U \in \mathcal{S}$. Hence, all thick ℓ-arrows leading to $L_{i,\ell,\mathcal{S}}$ and its copies have to be passive. To resolve all these $(q+1)$-critical graphs, we have to spend one extra unit of budget per copy which is not accounted for by the packing. Due to the large number of copies, this implies that we must violate our budget constraint.

Color-Set-Gadgets. The next step is to construct gadgets that can distinguish between the different partial solutions in Φ_i. To do so, these gadgets must be able to react not only based on the number of deletions inside a twinclass, but also based on the set of colors used for a twinclass. In Lemma 13.3.5, we construct *color-set-gadgets* $S_C(U,v)$ using LIST COLORING constraints simulated by the central clique F. Given a solution φ and a set of colors $C \subsetneq [q]$, the color-set-gadget $S_C(U,v)$ propagates a deletion to the vertex v when the set of colors $\varphi(U) \setminus \{\perp\}$ used on a set of true twins U is a subset of C, i.e., $\varphi(U) \setminus \{\perp\} \subseteq C$. We say that $S_C(U,v)$ is *active* if a deletion is propagated to v and *passive* otherwise.

Lemma 13.3.5. *Let U be a set consisting of at most q true twins and v be a vertex that is not adjacent to U and let $C \subsetneq [q]$ with $|C| \leq |U|$. There is a graph $S = S_C(U, v)$ with the following properties:*

- *Any solution φ, also unnormalized ones, satisfies $|V(S-U) \cap \varphi^{-1}(\bot)| \geq (q - |C|) + 1$.*
- *If φ is a normalized solution with $|V(S-U) \cap \varphi^{-1}(\bot)| = (q-|C|)+1$, then $\varphi(v) = \bot$ implies that $\varphi(U) \setminus \{\bot\} \subseteq C$.*
- *Any normalized solution $\tilde{\varphi}$ for $\tilde{G} = G[(V \setminus V(S)) \cup U \cup \{v\}]$ with $\tilde{\varphi}(v) \neq \bot$ or $\tilde{\varphi}(U) \setminus \{\bot\} \subseteq C$ can be extended to a normalized solution φ for G with $|V(S-U) \cap \varphi^{-1}(\bot)| = (q-|C|)+1$.*
- *$\mathrm{tw}(S - U) \leq q$ via a winning cops-and-robber-strategy starting with a cop on v.*

Proof. Let $[q] \setminus C = \{\overline{c}_1, \ldots, \overline{c}_\ell\}$. We add $2\ell + 1$ vertices $w_1, \ldots, w_{2\ell+1}$ and add all edges between w_{2i-1}, $i \in [\ell]$, and U, all edges between w_{2i}, $i \in [\ell]$, and $w_{2\ell+1}$, a deletion edge between $w_{2\ell+1}$ and v and a deletion edge between each pair w_{2i-1} and w_{2i} for $i \in [\ell]$. We also add all edges between $\{w_{2i} : i \in [\ell]\} \cup \{w_{2\ell+1}\}$ and $F \setminus \{f_1\}$. Finally, for each $i \in [\ell]$, vertex w_{2i-1} is adjacent to all vertices in $F \setminus \{f_{\overline{c}_i}\}$. Hence, considering a normalized solution, the vertices in $\{w_{2i} : i \in [\ell]\} \cup \{w_{2\ell+1}\}$ may only receive color 1 or be deleted and vertex w_{2i-1}, $i \in [\ell]$, may only receive color $\overline{c}_i$ or be deleted. See Figure 13.7 for a depiction of the construction.

The lower bound on the number of deletions for possibly unnormalized solutions follows by noticing that $S - U$ contains $\ell + 1 = (q - |C|) + 1$ disjoint deletion edges.

Suppose that φ is a normalized solution with $|V(S-U) \cap \varphi^{-1}(\bot)| = \ell+1$ and $\varphi(U) \setminus \{\bot\} \not\subseteq C$, then there exists some $\overline{c}_i \in ([q] \setminus C) \cap \varphi(U)$. Due to the constraints enforced by F, we must have that $\varphi(w_{2i-1}) = \bot$ and $\varphi(w_{2i}) = 1$ by the bound on the number of deletions. This in turn forces $\varphi(w_{2\ell+1}) = \bot$ and hence v cannot be deleted without violating the deletion bound.

Suppose that $\tilde{\varphi}$ is a normalized solution of $\tilde{G}$ with $\tilde{\varphi}(v) \neq \bot$, then we can extend to a normalized solution φ of G by deleting all vertices in $\{w_{2i-1} : i \in [\ell]\} \cup \{w_{2\ell+1}\}$. It is easy to see that the remainder can be colored correctly.

Suppose that $\tilde{\varphi}$ is a normalized solution of $\tilde{G}$ with $\tilde{\varphi}(U) \setminus \{\bot\} \subseteq C$ and $\tilde{\varphi}(v) = \bot$. Since $([q] \setminus C) \cap \tilde{\varphi}(U) = \emptyset$, we can extend $\tilde{\varphi}$ to a normalized solution φ of G by setting $\varphi(w_{2i-1}) = \overline{c}_i$, $\varphi(w_{2i}) = \bot$ for all $i \in [\ell]$, and $\varphi(w_{2\ell+1}) = 1$. It is easy to see that the remainder can be colored correctly.

We argue that $\mathrm{tw}(S - U) \leq q$ using the omniscient cops-and-robber-game. We begin by placing cops on v and $w_{2\ell+1}$. This splits the graph into $\ell + 1$ connected components, one per deletion edge. We place two cops on the endpoints of the deletion edge the robber escaped to. If v or respectively $w_{2\ell+1}$ is not an endpoint of the considered deletion edge, then we remove the cop from v or respectively $w_{2\ell+1}$. Finally, we place $q - 1$ cops on the vertices inside the deletion edge and capture the robber. This proves the treewidth bound by Theorem 2.4.30. □

In the case of $C = \emptyset$ one could also use a thick q-arrow from Lemma 13.3.2 instead. We will only invoke Lemma 13.3.5 with $|U| = q$ in the dense setting of the DELETION TO q-COLORABLE lower bound; we use the case $|U| = 1$ for the sparse version of the lower bound.

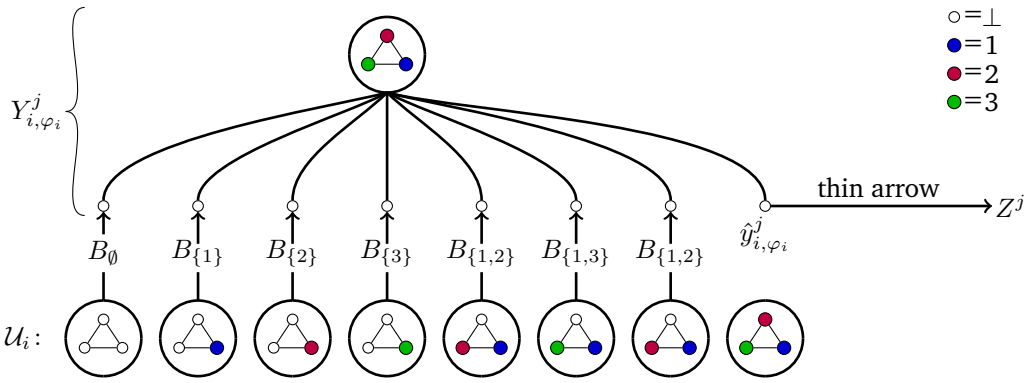


Fig. 13.8.: The decoding gadget Y^j_{i,φ_i} and its connections to the central twinclasses and the clause gadget for the case $q = 3$ and $p = 8$. The large circles represent twinclasses and the arrows represent color-set-gadgets or thin arrows. The solution chosen on the central twinclasses is φ_i. Note that it is allowed that φ_i uses the same color set on several twinclasses in $\mathcal{U}_i$.

Decoding Gadgets. The color-set-gadgets allow us to construct *decoding gadgets* that can distinguish between the different partial solutions in Φ_i. While the color-set-gadgets only check for inclusion and not equality, this is nonetheless sufficient to distinguish solutions in Φ_i due to their structure.

We begin with the construction now. For the j-th clause, group $i \in [t]$, solution $\varphi_i \in \Phi_i$, we construct a gadget Y^j_{i,φ_i} as follows. The gadget Y^j_{i,φ_i} consists of a large independent set joined to a K_q, i.e., adding all edges between both sets. In other words, Y^j_{i,φ_i} is a complete $(q+1)$-partite graph with one large independent set and all other independent sets in the partition are singletons. More precisely, the large independent set consists of $(1-2^{-q})p+1$ vertices (recall that 2^q divides p). One of these vertices is distinguished and denoted by $\hat{y}^j_{i,\varphi_i}$. This concludes the construction, see Figure 13.8 for a depiction of the construction.

In particular, the K_q and $\hat{y}^j_{i,\varphi_i}$ induce a complete graph of size $q+1$. Hence, any solution must delete at least one vertex in this complete graph, and due to the budget constraint exactly one vertex has to be deleted. The distinguished vertex $\hat{y}^j_{i,\varphi_i}$ can only be deleted if also all other vertices in the large independent set are deleted. By appropriately adding color-set-gadgets, we will ensure that this can only be achieved if φ_i is chosen on $\mathcal{U}_i$.

For the j-th clause, group $i \in [t]$, solution $\varphi_i \in \Phi_i$, twinclass $U \in \mathcal{U}_i$ with $\varphi_i(U) \neq [q]$, we pick a private vertex $v \neq \hat{y}^j_{i,\varphi_i}$ in the large independent set of Y^j_{i,φ_i} and add the color-set-gadget $S_{\varphi_i(U)\setminus\{\perp\}}(U,v)$ and denote this instance of the color-set-gadget by $W^j_{i,\varphi_i,U}$. We will see, using the properties of solutions in Φ_i, that if the solution on $\mathcal{U}_i$ diverges from φ_i, then at least one $W^j_{i,\varphi_i,v}$ will be passive. Otherwise, all $W^j_{i,\varphi_i,v}$ will be active, allowing us to delete the vertex $\hat{y}^j_{i,\varphi_i}$ in Y^j_{i,φ_i}.

Clause Gadgets. For the j-th clause, we add an $(q+1)$-critical graph, denoted $Z^j \in \mathcal{H}^q$, consisting of at least $n2^\beta$ vertices. For every group $i \in [t]$ and solution $\varphi_i \in \Phi_i$ such that $\kappa_i^{-1}(\varphi_i)$ is a partial truth assignment satisfying the j-th clause, we pick a private vertex v in Z^j and add a thin arrow from $\hat{y}^j_{i,\varphi_i}$ to v. Since each variable group has size at most β, there are at most 2^β partial truth assignments per variable group that satisfy the j-th

clause. Therefore, Z^j contains enough vertices so that we can pick a private one for every considered partial truth assignment. This concludes the construction of the clause gadget.

The idea of the clause gadget is that we can only afford to delete a vertex in Z^j if we have picked a partial solution on some $\mathcal{U}_i$ that corresponds to a satisfying truth assignment of the j-th clause, which will propagate a deletion to Z^j via the decoding gadgets.

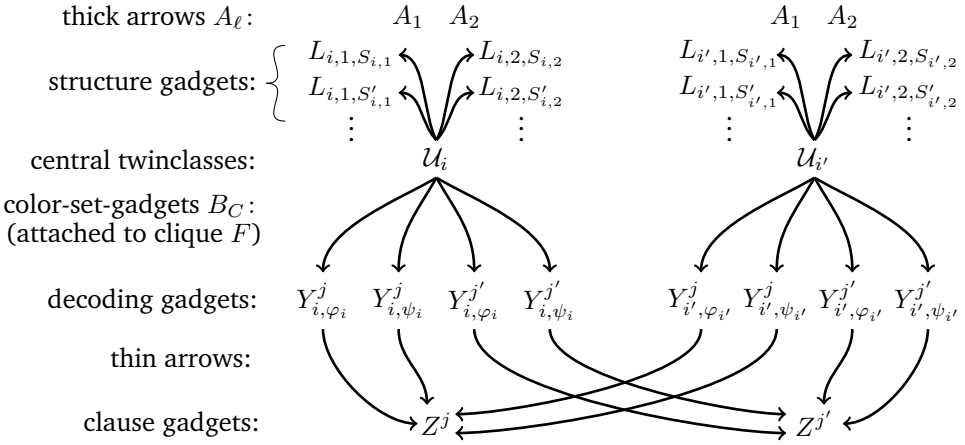


Fig. 13.9.: An overview of the construction of the graph $G(\sigma, q, \beta)$ for the case $q = 2$.

Packing. We will now construct a packing $\mathcal{P}$ of vertex-disjoint graphs that will fully explain the budget outside of the central vertices. For every thin arrow from u to v in the construction, we add the K_{q+1} induced by the deletion edge incident to v to the packing $\mathcal{P}$. For every thick ℓ-arrow $A_\ell(U, v)$ in the construction, we add $A_\ell(U, v) - U$ to the packing $\mathcal{P}$ and by Lemma 13.3.2 these graphs require at least ℓ deletions each. For every color-set-gadget $S_C(U, v)$ in the construction, we add $S_C(U, v) - U$ to the packing $\mathcal{P}$ and by Lemma 13.3.5 these graphs require at least $(q - |C|) + 1$ deletions each. Finally, for every Y^j_{i,φ_i}, we add the K_{q+1} induced by $\hat{y}^j_{i,\varphi_i}$ and the K_q of Y^j_{i,φ_i} to the packing $\mathcal{P}$. Let $\mathrm{cost}_{\mathcal{P}}$ denote the cost of the packing $\mathcal{P}$.

Observe that no vertex will be the head of several arrows or color-set-gadgets in our construction, hence the graphs in $\mathcal{P}$ are indeed vertex-disjoint. Furthermore, no graph in $\mathcal{P}$ intersects the central vertices and the cost of $\mathcal{P}$ is independent of the partial solution chosen on the central vertices. Finally, notice that the cost of $S_C(U, v) - U$ is fully explained by the $(q - |C|) + 1$ disjoint deletion edges and hence does not rely on any List Coloring constraints simulated by the central clique F.

This concludes the construction of $G = G(\sigma, q, \beta)$, cf. Figure 13.9. We proceed by showing the correctness of the reduction. From now on, we might omit the range of the indices for the sake of readability, but we keep the meaning of the indices consistent throughout. By i we denote a group of variables, φ_i denotes a partial solution in Φ_i, and U denotes a true twinclass in $\mathcal{U}_i$ with $\varphi_i(U) \neq [q]$.

13.3.3 Proofs

Theorem 13.3.6. *Let σ be a* SATISFIABILITY *instance with n variables and m clauses. Let $G = G(\sigma, q, \beta)$ and $\mathcal{P}$ be the graph and packing as constructed above and let $\bar{b} = \mathrm{cost}_{\mathcal{P}} + tqp/2$. If σ has a satisfying assignment τ, then there is a solution φ of the* DELETION TO q-COLORABLE *instance $(G, \bar{b})$.*

Proof. We start with the construction of φ on the central vertices. We color the central clique F with $\varphi(f_s) = s$ for $s \in [q]$, i.e., φ will be a normalized solution. For each group i of variables, let $\tau_i \in \{0,1\}^\beta$ be the partial truth assignment on group i induced by τ. For each group i, we set $\varphi\big|_{\bigcup(\mathcal{U}_i)} = \varphi_i := \kappa_i(\tau_i) \in \Phi_i$. By Lemma 13.3.3, this results in exactly $tqp/2$ deletions. Hence, only the budget $\mathrm{cost}_{\mathcal{P}}$ for the packing $\mathcal{P}$ remains, so for every graph in $\mathcal{P}$ we have to match the lower bound on the number of deletions for this graph.

Whenever we delete the tail u or head v of a thin arrow from u to v, then we use the active solution on this arrow; otherwise, we use the passive solution. Either type of solution results in exactly one deletion in the deletion edge incident to v which belongs to $\mathcal{P}$.

Similarly, for any thick ℓ-arrow $A_\ell(U, v)$, we extend to the solution with $|V(A_\ell(U,v) - U) \cap \varphi^{-1}(\bot)| = \ell$ as given by Lemma 13.3.2. If $|U \cap \varphi^{-1}(\bot)| \geq \ell$, then this solution is active and satisfies $\varphi(v) = \bot$.

For every color-set-gadget $S_C(U, v)$, we extend to the solution with $|V(S_C(U,v) - U) \cap \varphi^{-1}(\bot)| = (q - |C|) + 1$ as given by Lemma 13.3.5. If $\varphi(U) \setminus \{\bot\} \subseteq C$, then this solution is active and satisfies $\varphi(v) = \bot$.

For every decoding gadget Y^j_{i,φ_i}, we delete the distinguished vertex $\hat{y}^j_{i,\varphi_i}$. For every j-th clause, group i, and solution $\psi_i \in \Phi_i \setminus \{\varphi_i\}$, we pick one of the vertices in the K_q of Y^j_{i,ψ_i} and delete it. This deletes exactly one vertex in the K_{q+1} formed by $\hat{y}^j_{i,\varphi_i}$ or $\hat{y}^j_{i,\psi_i}$ and the K_q.

This concludes the description of the deletions; for each graph in the packing $\mathcal{P}$, we match the lower bound on the number of deletions for this graph. It remains to show that the remainder of G can be properly q-colored. This can be easily seen for the thin arrows. For thick ℓ-arrows $A_\ell(U, v)$ it follows from Lemma 13.3.2. For the color-set-gadgets $W^j_{i,\psi_i,U}$, where $\psi_i \in \Phi_i$, it follows from Lemma 13.3.5. It remains to handle the structure gadgets $L_{i,\ell,S}$, the decoding gadgets Y^j_{i,ψ_i}, $\psi_i \in \Phi_i$, and the clause gadgets Z^j.

By Lemma 13.3.2 and Lemma 13.3.5, we can appropriately extend solutions to $A_\ell(U, v)$ and $S_C(U, v)$ for every color choice on the vertex v. Hence, when handling the aforementioned gadgets, it does not matter how we color the undeleted vertices. It is sufficient to verify that enough deletions occur and that the remainder of the gadget can be q-colored.

Consider some group i, some $\ell \in [q]$, some $\mathcal{S} \subseteq \mathcal{U}_i$ with $|\mathcal{S}| = \binom{q}{\geq(q-\ell+1)}\frac{p}{2^q} + 1$, and some copy of $L_{i,\ell,\mathcal{S}}$. Since $L_{i,\ell,\mathcal{S}}$ is an $(q+1)$-critical graph that does not belong to $\mathcal{P}$, we must perform at least one deletion in $L_{i,\ell,\mathcal{S}}$ and this deletion has to be propagated to $L_{i,\ell,\mathcal{S}}$ by some thick ℓ-arrow $A_\ell(U, v)$, $U \in \mathcal{U}_i$. Since $\varphi_i \in \Phi_i$, we have that $\left|\left\{U \in \mathcal{U}_i : \varphi_i(U) \setminus \{\bot\} \in \binom{[q]}{\leq(q-\ell)}\right\}\right| = \binom{q}{\leq(q-\ell)}\frac{p}{2^q}$, i.e., there are exactly $\binom{q}{\leq(q-\ell)}\frac{p}{2^q}$ twinclasses in $\mathcal{U}_i$ where φ_i deletes at least ℓ vertices in each. Due to $\binom{q}{\leq(q-\ell)}\frac{p}{2^q} + |\mathcal{S}| = p + 1 > p = |\mathcal{U}_i|$, there must be at least one twinclass $U \in \mathcal{S}$ with $\varphi_i(U) \setminus \{\bot\} \in \binom{[q]}{\leq(q-\ell)}$, i.e., where φ_i deletes at least ℓ vertices. Therefore, the thick ℓ-arrow $A_\ell(U, v)$ leading to this

copy of $L_{i,\ell,\mathcal{S}}$ propagates a deletion to $L_{i,\ell,\mathcal{S}}$ by Lemma 13.3.2 and thereby resolves this $(q+1)$-critical graph.

Consider some Y^j_{i,ψ_i}, where $\psi_i \in \Phi_i$, if $\psi_i \neq \varphi_i$, then one of the vertices in the K_q is deleted and this $(q+1)$-partite graph is resolved. If $\psi_i = \varphi_i$, then we claim that the large independent set is fully deleted. In the construction of φ, we distributed a deletion to $\hat{y}^j_{i,\varphi_i}$. All other vertices v of the large independent set are hit by some $W^j_{i,\varphi_i,U}$, where $\varphi_i(U) \setminus \{\bot\} \neq [q]$. By Lemma 13.3.5, we see that $W^j_{i,\varphi_i,U} = S_{\varphi_i(U)\setminus\{\bot\}}(U, v)$ propagates a deletion to v due to Lemma 13.3.5. Due to $\varphi_i \in \Phi_i$, there are $\binom{q}{\leq(q-1)}\frac{p}{2^q} = (1-2^{-q})p$ twinclasses $U \in \mathcal{U}_i$ with $\varphi_i(U) \setminus \{\bot\} \neq [q]$, thus matching the size of the large independent set of Y^j_{i,φ_i} with the exception of $\hat{y}^j_{i,\varphi_i}$. This shows that the large independent set is fully deleted and the $(q+1)$-partite graph Y^j_{i,φ_i} is resolved.

Finally, consider some Z^j, then there is some group i such that τ_i satisfies the j-th clause because τ is a satisfying assignment of σ. By construction $\varphi_i = \kappa_i(\tau_i)$ and the vertex $\hat{y}^j_{i,\varphi_i}$ is deleted, so the thin arrow from $\hat{y}^j_{i,\varphi_i}$ to Z^j is active and propagates a deletion to Z^j. Therefore, the $(q+1)$-critical graph Z^j is resolved as well. □

Theorem 13.3.7. *Let σ be a* SATISFIABILITY *instance with n variables and m clauses. Let $G = G(\sigma, q, \beta)$ be the graph as constructed above and let $\bar{b} = \mathrm{cost}_{\mathcal{P}} + tqp/2$. If φ is a solution of the* DELETION TO q-COLORABLE *instance $(G, \bar{b})$, then $|\varphi^{-1}(\bot)| = \bar{b}$. Furthermore, there is a normalized solution ψ with $|\psi^{-1}(\bot)| = |\varphi^{-1}(\bot)| = \bar{b}$ and $\psi|_{\bigcup(\mathcal{U}_i)} \in \Phi_i$ for all $i \in [t]$.*

Proof. As argued in the construction of $\mathcal{P}$, the packing $\mathcal{P}$ forces φ to spend at least $\mathrm{cost}_{\mathcal{P}}$ units of budget outside of the central vertices. We claim that for any group $i \in [t]$ and number of deletions $\ell \in [q]$, there must be at least $\binom{q}{\leq(q-\ell)}\frac{p}{2^q}$ twinclasses $U \in \mathcal{U}_i$ such that $|U \cap \varphi^{-1}(\bot)| \geq \ell$, i.e., φ deletes at least ℓ vertices in each of those U. We first define $\mathcal{S}^\ell_i(\varphi) = \{U \in \mathcal{U}_i : |U \cap \varphi^{-1}(\bot)| = \ell\}$ and $\mathcal{S}^{\geq\ell}_i(\varphi) = \mathcal{S}^\ell_i(\varphi) \cup \mathcal{S}^{\ell+1}_i(\varphi) \cup \cdots \cup \mathcal{S}^q_i(\varphi)$ for all $i \in [t]$ and $\ell \in [0, q]$.

Suppose that there is some group i and some number of deletions $\ell \in [q]$ with $|\mathcal{S}^{\geq\ell}_i(\varphi)| < \binom{q}{\leq(q-\ell)}\frac{p}{2^q}$. Then there exists a family $\mathcal{S} \subseteq \mathcal{U}_i$ of twinclasses with $|\mathcal{S}| = \binom{q}{\geq(q-\ell+1)}\frac{p}{2^q} + 1$ and $\mathcal{S}^{\geq\ell}_i(\varphi) \cap \mathcal{S} = \emptyset$. Consider any copy of $L_{i,\ell,\mathcal{S}}$ and the thick ℓ-arrows $A_\ell(U, v)$ from $\mathcal{S} \subseteq \mathcal{U}_i$ to $L_{i,\ell,\mathcal{S}}$. Due to $\mathcal{S}^{\geq\ell}_i(\varphi) \cap \mathcal{S} = \emptyset$, the solution φ deletes at most $\ell - 1$ vertices in each $U \in \mathcal{S}$, and by Lemma 13.3.2 this implies that no vertex in the copy of $L_{i,\ell,\mathcal{S}}$ is deleted, unless we pay for at least one extra deletion not accounted for by the packing $\mathcal{P}$ per copy of $L_{i,\ell,\mathcal{S}}$. However, since $L_{i,\ell,\mathcal{S}}$ is an $(q+1)$-critical graph, we must delete at least one vertex in each copy. This would exceed our remaining budget, because there are $1 + tqp/2 > \bar{b} - \mathrm{cost}_{\mathcal{P}}$ copies of $L_{i,\ell,\mathcal{S}}$. Hence, we must have that $|\mathcal{S}^{\geq\ell}_i(\varphi)| \geq \binom{q}{\leq(q-\ell)}\frac{p}{2^q}$ for all groups $i \in [t]$ and deletions $\ell \in [q]$.

We now argue that $|\mathcal{S}_i^\ell(\varphi)| = \binom{q}{q-\ell}\frac{p}{2^q}$ for all $i \in [t]$ and $\ell \in [q]$ due to our budget constraint. Consider some group i, using the previous inequalities we can derive the following bound on the total number of deletions performed on $\mathcal{U}_i$ by φ:

$$\sum_{\ell=1}^{q} \ell|\mathcal{S}_i^\ell(\varphi)| = q|\mathcal{S}_i^q(\varphi)| + \sum_{\ell=1}^{q-1} \ell(|\mathcal{S}_i^{\geq \ell}(\varphi)| - |\mathcal{S}_i^{\geq(\ell+1)}(\varphi)|) = \sum_{\ell=1}^{q} |\mathcal{S}_i^{\geq \ell}(\varphi)|$$
$$\geq \sum_{\ell=1}^{q} \binom{q}{\leq (q-\ell)}\frac{p}{2^q} = \sum_{\ell=0}^{q-1}(q-\ell)\binom{q}{\ell}\frac{p}{2^q} = qp/2,$$

where the last inequality follows from the computation in the proof of Lemma 13.3.3. Summing over the lower bound for all groups, we see that this uses up the whole remaining budget and hence the inequality must be tight for every group. Therefore, also $|\mathcal{S}_i^{\geq \ell}(\varphi)| = \binom{q}{\leq (q-\ell)}\frac{p}{2^q}$ and $|\mathcal{S}_i^\ell(\varphi)| = |\mathcal{S}_i^{\geq \ell}(\varphi)| - |\mathcal{S}_i^{\geq(\ell+1)}(\varphi)| = \binom{q}{q-\ell}\frac{p}{2^q}$ for all $i \in [t]$ and $\ell \in [q]$. Since $\sum_{\ell=0}^{q} |\mathcal{S}_i^\ell(\varphi)| = p$, this also implies that $|\mathcal{S}_i^0(\varphi)| = \binom{q}{q}\frac{p}{2^q} = \frac{p}{2^q}$ for all groups i.

No vertex of the central clique F can be deleted as φ spends its whole budget on the packing $\mathcal{P}$ and the families of twinclasses $\mathcal{U}_i$, $i \in [t]$. Hence, we can assume by permuting the colors that $\varphi(f_s) = s$, $s \in [q]$, i.e., φ is a normalized solution.

Finally, the equations $|\mathcal{S}_i^\ell(\varphi)| = \binom{q}{q-\ell}\frac{p}{2^q} = \binom{q}{\ell}\frac{p}{2^q}$, $i \in [t]$, $\ell \in [0, q]$, imply that for every group i there exists some $\psi_i \in \Phi_i$ with $\varphi(U) = \psi_i(U)$ for all $U \in \mathcal{U}_i$. Since all $U \in \mathcal{U}_i$ are twinclasses, exchanging $\varphi|_{\bigcup(\mathcal{U}_i)}$ with ψ_i for every group i still yields a solution ψ for DELETION TO q-COLORABLE with the desired properties. □

Theorem 13.3.8. *Let σ be a* SATISFIABILITY *instance with n variables and m clauses. Let $G = G(\sigma, q, \beta)$ be the graph as constructed above and let $\bar{b} = \mathrm{cost}_{\mathcal{P}} + tqp/2$. If the* DELETION TO q-COLORABLE *instance $(G, \bar{b})$ has a solution φ, then σ has a satisfying assignment τ.*

Proof. By Theorem 13.3.7, we know that $|\varphi^{-1}(\bot)| = \bar{b}$ and can assume that φ is a normalized solution with $\varphi_i := \varphi|_{\bigcup(\mathcal{U}_i)} \in \Phi_i$ for all groups i. Hence, the deletions are fully explained by the packing $\mathcal{P}$ and the properties of solutions in Φ_i.

For every thin arrow from u to v, thick ℓ-arrow $A_\ell(U, v)$, color-set-gadget $S_C(U, v)$, by Lemma 13.3.2 and Lemma 13.3.5, we can assume that φ chooses the active solution on this gadget if the appropriate condition at u or U is satisfied.

Consider the j-th clause and some group i, we argue that $\varphi(\hat{y}^j_{i,\psi_i}) = \bot$ can only hold if $\psi_i = \varphi_i$. Observe that $\hat{y}^j_{i,\psi_i}$ may only be deleted if all other vertices of the large independent set in Y^j_{i,ψ_i} are deleted, otherwise Y^j_{i,ψ_i}, which induces a complete $(q+1)$-partite graph, is not resolved or we exceed the packing budget on the complete graph induced by $\hat{y}^j_{i,\psi_i}$ and the K_q. Consider some $\psi_i \in \Phi_i \setminus \{\varphi_i\}$. We distinguish three cases based on the sizes of the color sets assigned to the twinclasses in $\mathcal{U}_i$ by φ_i and ψ_i.

1. Suppose that there is some $U \in \mathcal{U}_i$ such that $|\psi_i(U) \setminus \{\bot\}| < |\varphi_i(U) \setminus \{\bot\}|$. This implies that $\psi_i(U) \setminus \{\bot\} \neq [q]$, hence the gadget $W^j_{i,\psi_i,U} = S_{\psi_i(U)\setminus\{\bot\}}(U, v)$, where v is in the large independent set of Y^j_{i,ψ_i}, exists. The cardinality inequality also implies that $\varphi_i(U) \setminus \{\bot\} \not\subseteq \psi_i(U) \setminus \{\bot\}$, so by Lemma 13.3.5 and the budget being tight on the graphs of the packing $\mathcal{P}$, we see that v cannot be deleted. Hence, $\hat{y}^j_{i,\psi_i}$ cannot be deleted either.

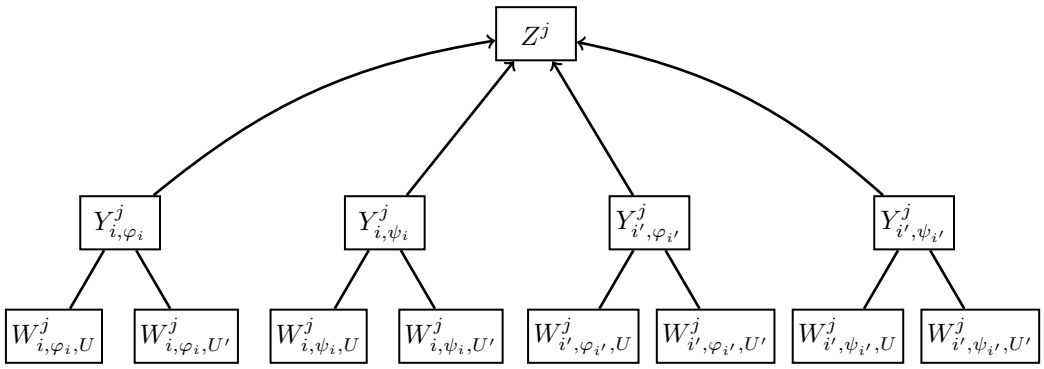


Fig. 13.10.: A high-level overview of the component corresponding to the j-th clause after removing the TCM $\mathcal{M}$. The arrows leading to Z^j denote thin arrows.

2. Suppose that there is some $U \in \mathcal{U}_i$ such that $|\psi_i(U) \setminus \{\bot\}| > |\varphi_i(U) \setminus \{\bot\}|$, then the first case must also apply for some $U' \in \mathcal{U}_i$, as otherwise ψ_i would perform too few deletions in $\mathcal{U}_i$ and hence $\psi_i \notin \Phi_i$.
3. Since the first two cases do not apply, it follows that $|\psi_i(U) \setminus \{\bot\}| = |\varphi_i(U) \setminus \{\bot\}|$ for all $U \in \mathcal{U}_i$. Due to $\psi_i \neq \varphi_i$, there must exist some $U \in \mathcal{U}_i$ such that $\psi_i(U) \setminus \{\bot\} \neq \varphi_i(U) \setminus \{\bot\}$. This implies that $\psi_i(U) \setminus \{\bot\} \neq [q]$ and hence the gadget $W^j_{i,\psi_i,U} = S_{\psi_i(U)\setminus\{\bot\}}(U, v)$, where v is in the large independent set of Y^j_{i,ψ_i}, exists. Since $|\psi_i(U) \setminus \{\bot\}| = |\varphi_i(U) \setminus \{\bot\}|$ and $\psi_i(U) \setminus \{\bot\} \neq \varphi_i(U) \setminus \{\bot\}$, we must have that $\varphi_i(U) \setminus \{\bot\} \not\subseteq \psi_i(U) \setminus \{\bot\}$, hence, as in the first case, v cannot be deleted and neither can $\hat{y}^j_{i,\psi_i}$.

This proves the claim regarding the deletion of $\hat{y}^j_{i,\psi_i}$.

Now, consider the gadget Z^j. To resolve Z^j, there has to be a thin arrow from some $\hat{y}^j_{i,\psi_i}$ to Z^j that is active since Z^j is not part of the packing $\mathcal{P}$. By the previous claim, this implies that $\psi_i = \varphi_i$. Furthermore, by construction of G such a thin arrow only exists if the partial truth assignment $\tau_i = \kappa_i^{-1}(\varphi_i)$ satisfies the j-th clause. Note that the definition of τ_i is independent of the considered clause.

For some groups i, the partial solution φ_i might not be in the image of κ_i, in that case τ_i is an arbitrary partial truth assignment for the i-th variable group. By the previous argument, the truth assignment $\tau = \bigcup_{i=1}^{t} \tau_i$ satisfies all clauses of σ. □

Theorem 13.3.9. *Let $G = G(\sigma, q, \beta)$ be the graph as constructed above. The set $\mathcal{M} = (\bigcup_{i=1}^{t} \mathcal{U}_i) \cup \{\{f_s\} : s \in [q]\}$ is a TCM for G of size $tp + q$ to treewidth q, i.e., $|\mathcal{M}| = tp + q$ and $\mathrm{tw}(G - \bigcup(\mathcal{M})) \leq q$.*

Proof. It follows from the construction of G that $\mathcal{M}$ has size $\sum_{i=1}^{t} |\mathcal{U}_i| + |F| = tp + q$ and only consists of twinclasses. The fact that all sets in $\mathcal{M}$ are twinclasses and not just sets of twins can be seen by considering the various color-set-gadgets $B_C(U, v)$ in the construction. It remains to argue that the treewidth of $G' := G - \bigcup(\mathcal{M})$ is at most q. We will show this by using the omniscient-cops-and-robber-game. The remaining graph G' consists of several connected components, namely one connected component per copy of $L_{i,\ell,\mathcal{S}}$ and one connected component for every clause. Observe that the heads of thick ℓ-arrows $A_\ell(U, v)$, of color-set-gadgets $S_C(U, v)$, of thin arrows, and also the tails of thin arrows consist of only a

single vertex. So, if the robber escapes through one of these gadgets and there is no other path back, then a single cop suffices to prevent the robber from going back.

We begin by handling the connected components corresponding to a copy of $L_{i,\ell,\mathcal{S}}$. On a high level, these components look like stars; the center is $L_{i,\ell,\mathcal{S}}$ and for every $U \in \mathcal{S}$ the remainder $A_\ell(U,v) - U$ of a thick ℓ-arrow is attached to $L_{i,\ell,\mathcal{S}}$. By Theorem 13.2.1, $L_{i,\ell,\mathcal{S}}$ has pathwidth q. With $q+1$ cops we sweep from left to right through the bags of the pathwidth decomposition of $L_{i,\ell,\mathcal{S}}$ until the robber escapes to one of the attached $A_\ell(U,v) - U$. When that happens we remove all cops, except the cop on the head v of the considered thick ℓ-arrow from $L_{i,\ell,\mathcal{S}}$. We can then capture the robber using the strategy given by Lemma 13.3.2.

Next, we handle the connected components corresponding to a clause. Figure 13.10 gives a high-level overview of how these components look, notice that at this level there are no cycles. Our strategy makes use of this hierarchy, we start at Z^j and chase the robber downwards, making sure that the robber cannot go back upwards. By Theorem 13.2.1, Z^j has pathwidth q. Again, we sweep with $q+1$ cops from left to right through the bags of the pathwidth decomposition of Z^j, until the robber escapes through some thin arrow coming from some Y^j_{i,φ_i}. Then, we remove all cops except the one on the head of the thin arrow and place one cop on the tail $\hat{y}^j_{i,\varphi_i}$ of the thin arrow. Either the robber escapes to the thin arrow, where we can capture it easily using $q-1$ further cops or the robber escapes to Y^j_{i,φ_i}.

If the robber escapes to Y^j_{i,φ_i}, then we remove the cop from the head of the thin arrow and place q cops on the K_q in Y^j_{i,φ_i}. This leaves us with one connected component $W^j_{i,\varphi_i,U} = S_C(U,v)$ for each $U \in \mathcal{U}_i$ with $\varphi_i(U) \neq [q]$. Consider the connected component the robber escaped to and move the cop from $\hat{y}^j_{i,\varphi_i}$ to v. Since there are still cops on the K_q in Y^j_{i,φ_i}, the robber cannot go back. We can now remove these cops on K_q and capture the robber using the strategy given by Lemma 13.3.5.

This concludes the strategy. Since we have never placed more than $q+1$ cops simultaneously, we see by Theorem 2.4.30 that G' has treewidth at most q. □

Proof of Theorem 13.3.1. Suppose there is some $q \geq 2$ and $\varepsilon > 0$ such that DELETION TO q-COLORABLE can be solved in time $\mathcal{O}^*((2^q - \varepsilon)^{|\mathcal{M}|})$, where $\mathcal{M}$ is a TCM to treewidth q. We will show that there exists a $\delta < 1$ such that we can solve SATISFIABILITY in time $\mathcal{O}(2^{\delta n}\operatorname{poly}(m)) = \mathcal{O}^*(2^{\delta n})$, where n is the number of variables and m the number of clauses. This contradicts CNF-SETH, Conjecture 2.1.1, and hence implies the desired lower bound.

Given a SATISFIABILITY instance σ, we construct the graph $G = G(\sigma, q, \beta)$, where β depends only on q and ε and will be chosen later. We can consider q and ε as constants, hence also $\beta = \beta(q,\varepsilon)$ and $p = p(\beta)$ are constant. First, we will argue that G has polynomial size. The number of vertices in the various gadgets can be estimated as follows, where we have an additional summand q for the $(q+1)$-critical graphs, since Theorem 13.2.1 only constructs them in increments of q:

- $|V(G[\bigcup(\mathcal{M})])| = \sum_{i=1}^{t} |\mathcal{U}_i|q + |F| = tpq + q = \lceil \frac{n}{\beta} \rceil pq + q = \mathcal{O}(n)$.
- $|V(L_{i,\ell,\mathcal{S}})| \leq |\mathcal{S}| + q \leq |\mathcal{U}_i| + q = p + q = \mathcal{O}(1)$.
- $|V(Z^j)| \leq n2^\beta + q = \mathcal{O}(n)$.
- $|V(Y^j_{i,\varphi_i})| \leq (1 - 2^{-q})p + 1 = \mathcal{O}(1)$.
- $|V(W^j_{i,\varphi_i,U})| \leq (q+1)^2 = \mathcal{O}(1)$.

- $|V(A_\ell(U,v))| \leq q^3 + 3q + 1 = \mathcal{O}(1)$.
- Every thin arrow consists of $2q+1$ vertices.

Next, we bound how often each gadget appears:

- i can take on $t = \lceil \frac{n}{\beta} \rceil = \mathcal{O}(n)$ values.
- As there are m clauses, j can take on m values.
- ℓ can take on $q = \mathcal{O}(1)$ values.
- For fixed i, $\mathcal{S} \subseteq \mathcal{U}_i$, so $\mathcal{S}$ can take on $2^{|\mathcal{U}_i|} = 2^p = \mathcal{O}(1)$ values.
- We create $1 + tqp/2 = \mathcal{O}(n)$ copies of each $L_{i,\ell,\mathcal{S}}$.
- For fixed i, $\varphi_i \in \Phi_i$ and $|\Phi_i| \leq (2^q)^p = \mathcal{O}(1)$.
- For fixed i, $U \in \mathcal{U}_i$, so U can take on $|\mathcal{U}_i| = p = \mathcal{O}(1)$ values.
- We create at most one (thin or thick) arrow per vertex in some Z^j or $L_{i,\ell,\mathcal{S}}$.

Together, all these bounds show that the size of G is polynomial in n, and m and following the construction of G, one can easily see that G can also be constructed in polynomial time. Furthermore, from the proof of Theorem 13.3.9 we can obtain a tree decomposition of $G' := G - \bigcup(\mathcal{M})$ of width q in polynomial time.

To analyze the running time resulting from applying the reduction and running the assumed algorithm for DELETION TO q-COLORABLE, we first bound p as follows:

$$p \leq \frac{\beta}{q} + 2^{q+1} \lceil \log_{2^q}(\beta/q) \rceil + 2^q + 1. \tag{13.1}$$

In the construction of $G = G(\sigma, q, \beta)$, we chose p as the smallest integer such that p is divisible by 2^q and such that the quantity from Lemma 13.3.4, which we denote by x, is larger than 2^β. The summand 2^q in (13.1) accounts for the divisibility. It remains to show that the second property is satisfied, for this, we work with $p = \frac{\beta}{q} + 2^{q+1}\lceil \log_{2^q}(\beta/q) \rceil + 1$.

We first observe that $(2^q-1)!/2^{(2^q)} \geq \frac{1}{4}$ for all $q \geq 2$. Hence, $x \geq (2^q)^p p^{-(2^q)}/4 =: x'$. Furthermore, observe that we have $2\beta/q \geq p$ for sufficiently large β. We proceed by showing that $x' \geq 2^\beta$:

$$\begin{aligned} x' &\geq (2^q)^{\frac{\beta}{q} + 2^{q+1}\lceil \log_{2^q}(\beta/q) \rceil} p^{-(2^q)} \geq 2^\beta \left(\frac{\beta}{q}\right)^{(2^{q+1})} p^{-(2^q)} \geq 2^\beta \left(\frac{\beta}{q}\right)^{(2^{q+1})} \left(2\frac{\beta}{q}\right)^{-(2^q)} \\ &\geq 2^\beta (\beta/q)^{(2^q)} 2^{-(2^q)} \geq 2^\beta, \end{aligned}$$

where we use $\beta/q \geq 2$ for sufficiently large β in the last inequality.

Running the assumed algorithm for DELETION TO q-COLORABLE on G decides, by Theorem 13.3.6 and Theorem 13.3.8, the satisfiability of σ. Since G can be constructed in polynomial time and has parameter value $|\mathcal{M}| = \lceil \frac{n}{\beta} \rceil p + q$ by Theorem 13.3.9, we can solve SATISFIABILITY in time

$$\begin{aligned} \mathcal{O}^*((2^q-\varepsilon)^{|\mathcal{M}|}) &= \mathcal{O}^*((2^q-\varepsilon)^{\lceil \frac{n}{\beta} \rceil p + q}) \leq \mathcal{O}^*((2^q-\varepsilon)^{\frac{n}{\beta}p}) \\ &\leq \mathcal{O}^*((2^q-\varepsilon)^{\frac{n}{q}} (2^q-\varepsilon)^{\frac{n}{\beta}(2^{q+1}\lceil \log_{2^q}(\beta/q) \rceil + 2^q + 2)}). \end{aligned}$$

We have that $\mathcal{O}^*((2^q-\varepsilon)^{\frac{n}{q}}) \leq \mathcal{O}^*(2^{\delta_1 n})$ for some $\delta_1 < 1$. It remains to upper bound the second factor, we will again use that β can be chosen sufficiently large:

$$\begin{aligned} & \mathcal{O}^*\left((2^q-\varepsilon)^{\frac{n}{\beta}(2^{q+1}\lceil \log_{2^q}(\beta/q)\rceil + 2^q+2)}\right) && \leq \mathcal{O}^*\left((2^q-\varepsilon)^{2^{q+2}\frac{n}{\beta}\log_{2^q}(\beta/q)}\right) \\ \leq\ & \mathcal{O}^*\left((2^q-\varepsilon)^{2^{q+2}\frac{n}{\beta}\log_{2^q}(\beta)}\right) && \leq \mathcal{O}^*\left(\left((2^q-\varepsilon)^{2^{q+2}\frac{\log_{2^q}(\beta)}{\beta}}\right)^n\right) \\ \leq\ & \mathcal{O}^*\left(2^{\delta_2 n}\right), \end{aligned}$$

where δ_2 can be chosen to be arbitrarily close to 0 by making β sufficiently large. By choosing δ_2 small enough so that $\delta := \delta_1 + \delta_2 < 1$, we obtain that SATISFIABILITY can be solved in time $\mathcal{O}^*(2^{\delta n})$, contradicting CNF-SETH, Conjecture 2.1.1. □

Corollary 13.3.10. *If* DELETION TO q-COLORABLE *can be solved in time* $\mathcal{O}^*((2^q-\varepsilon)^{\text{tc-td}(G)})$ *for some* $q \geq 2$ *and* $\varepsilon > 0$*, then CNF-SETH is false.*

Proof. Follows from Theorem 13.3.1 and Lemma 2.4.23. □

13.4 Sparse Setting

We again view *solutions* to DELETION TO q-COLORABLE as functions $\varphi\colon V(G) \to [q] \cup \{\perp\}$ with the property discussed in the outline, cf. Section 13.1.

In this subsection, we show how to adapt the lower bound for the dense setting to the sparse setting. We will give the construction in full detail again, but since the principle of construction is so similar we will only explain how to adapt the previous proofs. One can see that the lower bound is tight by a routine application of dynamic programming on tree decompositions.

Theorem 13.4.1. *If* DELETION TO q-COLORABLE *can be solved in time* $\mathcal{O}^*((q+1-\varepsilon)^{|M|})$ *for some* $q \geq 2$ *and* $\varepsilon > 0$*, where* M *is a modulator to treewidth* q*, then CNF-SETH is false.*

The remainder of this section is devoted to proving Theorem 13.4.1. Assume that we can solve DELETION TO q-COLORABLE in time $\mathcal{O}^*((q+1-\varepsilon)^{|M|})$ for some $q \geq 2$ and $\varepsilon > 0$. We provide a reduction from SATISFIABILITY with n variables and m clauses to DELETION TO q-COLORABLE with a modulator to treewidth q of size approximately $n \log_{q+1}(2)$ and graph size polynomial in n and m. Together with the assumed faster algorithm, this implies a faster algorithm for SATISFIABILITY, thus violating CNF-SETH, Conjecture 2.1.1. We will consider q to be fixed from now on.

Construction. Consider a SATISFIABILITY instance σ with n variables and m clauses. We enumerate the clauses and refer to them by their number. We pick an integer β which only depends on ε and q; we will describe how to choose β later. We partition the variables of σ into groups of size at most β, resulting in $t = \lceil n/\beta \rceil$ groups which will be indexed by i. Next, we choose the smallest integer p such that p is divisible by $q+1$ and $(q+1)^p \frac{p!q!}{(p+q)!} \geq 2^\beta$. We will now describe the construction of the DELETION TO q-COLORABLE instance $G = G(\sigma, q, \beta)$.

Comparison to Dense Setting. The principle behind the construction is essentially the same as for the dense setting, cf. Section 13.3, but the gadgets can be simplified. The central twinclasses will simply be single vertices now. Hence, we will not have to distinguish between different numbers of deletions in a twinclass, instead we simply distinguish between whether a vertex is deleted or not. Most notably, this allows us to use thin arrows instead of thick ℓ-arrows and the structure of the considered partial solutions on the central vertices simplifies significantly. The remaining gadgets structurally stay the same, but their size may change.

Construction of Central Vertices. The central vertices of G form the modulator to treewidth q. For each variable group $i \in [t]$, we create an independent set U_i consisting of p vertices. Furthermore, we again have a central clique $F = \{f_s : s \in [q]\}$ consisting of q vertices that is used to simulate LIST COLORING constraints. A solution $\varphi\colon V \to [q] \cup \{\bot\}$ satisfying $\varphi^{-1}(\bot) \cap F = \emptyset$ and $\varphi(f_s) = s$ for all $s \in [q]$ is called a *normalized solution*.

As described in the outline, we only want to consider solutions that delete a fixed number of vertices per group U_i. By using slightly more vertices p than enforced by the base conversion, sufficiently many solutions remain, if we define Φ_i as follows.

Definition 13.4.2. The family Φ_i consists of all $\varphi\colon U_i \to [q] \cup \{\bot\}$ that satisfy the equation $|\varphi^{-1}(\bot)| = p/(q+1)$.

Lemma 13.4.3. *We have that* $|\Phi_i| \geq (q+1)^p \frac{p!q!}{(p+q)!}$ *for all* $i \in [t]$.

Proof. Since Φ_i contains all $\varphi\colon U_i \to [q] \cup \{\bot\}$ with $|\varphi^{-1}(c)| = p/(q+1)$ for all $c \in [q] \cup \{\bot\}$, we see that $|\Phi_i| \geq \binom{p}{\frac{p}{q+1},\ldots,\frac{p}{q+1}} = x$, where x is the central multinomial coefficient. It can be seen that x is a maximum of the function $(c_1, \ldots, c_{q+1}) \mapsto \binom{p}{c_1,\ldots,c_{q+1}}$. The number of summands in the multinomial theorem $\sum_{c_1+\cdots c_{q+1}=p} \binom{p}{c_1,\ldots,c_{q+1}} = (q+1)^p$ is $\binom{p+q}{p} = \frac{(p+q)!}{p!q!}$, which corresponds to the number of weak compositions of p into $q+1$ parts, and x is one of them. Hence, we see that $|\Phi_i| \geq x \geq (q+1)^p \frac{p!q!}{(p+q)!}$. □

Hence, by the choice of p, we can pick for each group $i \in [t]$ an efficiently computable injective mapping $\kappa_i\colon \{0,1\}^\beta \to \Phi_i$ that maps truth assignments of the i-th variable group to partial solutions on U_i with the desired structure.

Budget. The budget $\bar{b} = |\mathcal{P}| + tp/(q+1)$ consists of two parts again. The first part $|\mathcal{P}|$ is allocated to a vertex-disjoint packing $\mathcal{P}$ of $(q+1)$-critical graphs and the second part $tp/(q+1)$ is allocated to the central vertices.

Enforcing Structure on Central Vertices. For every group $i \in [t]$ and subset $S \subseteq U_i$ with $|S| = \frac{q}{q+1}p + 1$, we add an $(q+1)$-critical graph, denoted $L_{i,S}$, consisting of at least $|S|$ vertices. For every $u \in S$, we pick a private vertex v in $L_{i,S}$ and add a thin arrow from u to v. We create $1 + tp/(q+1) = 1 + (\bar{b} - |\mathcal{P}|)$ copies of $L_{i,S}$ and the incident arrows. This concludes the construction of the structure gadget.

Decoding Gadgets. For the j-th clause, group $i \in [t]$, partial solution $\varphi_i \in \Phi_i$, we construct a decoding gadget Y^j_{i,φ_i} as follows. The gadget Y^j_{i,φ_i} consists of a large independent set joined to a K_q, i.e., adding all edges between both sets. In other words, Y^j_{i,φ_i} is a complete $(q+1)$-partite graph with one large independent set, and all other independent sets in the partition are singletons. The large independent set consists of $p+1$ vertices, where one of these vertices is distinguished and denoted by $\hat{y}^j_{i,\varphi_i}$. This concludes the construction of Y^j_{i,φ_i}.

For the j-th clause, group $i \in [t]$, partial solution $\varphi_i \in \Phi_i$, vertex $u \in U_i$, we pick a private vertex $v \neq \hat{y}^j_{i,\varphi_i}$ in the large independent set of Y^j_{i,φ_i} and add the decoding gadget $S_{\{\varphi_i(u)\}\setminus\{\perp\}}(\{u\}, v)$ and denote this instance of the decoding gadget by $W^j_{i,\varphi_i,u}$. Observe that in the dense setting no twinclasses U with $\varphi_i(U) \neq [q]$ have received decoding gadgets, whereas there is no such exception in the sparse setting.

Clause Gadgets. For the j-th clause, we add an $(q+1)$-critical graph, denoted Z^j, consisting of at least $n2^\beta$ vertices. For every group $i \in [t]$ and partial solution $\varphi_i \in \Phi_i$ such that $\kappa^{-1}(\varphi_i)$ is a partial truth assignment satisfying the j-th clause, we pick a private vertex v in Z^j and add a thin arrow from $\hat{y}^j_{i,\varphi_i}$ to v. We ensured that Z^j is large enough so that we can always pick such a private vertex. This concludes the construction of $G(\sigma, q, \beta)$, cf. Figure 13.11.

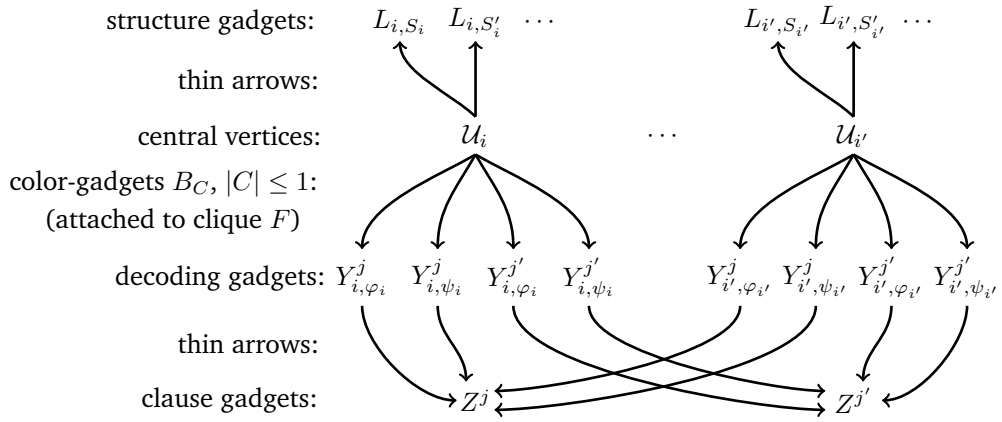


Fig. 13.11.: An overview of the construction of the graph $G(\sigma, q, \beta)$ for the case $q = 2$.

Packing. We construct a vertex-disjoint packing $\mathcal{P}$ of $(q+1)$-critical graphs that fully explains the budget outside of the central vertices. For every thin arrow from u to v in the construction, we add the K_{q+1} induced by the deletion edge incident to the head v of the arrow to the packing $\mathcal{P}$. Every $S_{\{\varphi_i(u)\}\setminus\{\perp\}}(\{u\}, v)$ contains, depending on whether $\varphi_i(u) \neq \perp$ or not, q or $q+1$ disjoint K_{q+1} corresponding to deletion edges, one of them being incident to v, and we add all these K_{q+1} to the packing $\mathcal{P}$. Finally, for every Y^j_{i,φ_i}, we add the K_{q+1} induced by $\hat{y}^j_{i,\varphi_i}$ and the K_q of Y^j_{i,φ_i} to the packing $\mathcal{P}$. Since, we have added only $(q+1)$-critical graphs to $\mathcal{P}$, the cost of $\mathcal{P}$ is simply $|\mathcal{P}|$.

Theorem 13.4.4. *Let σ be a* SATISFIABILITY *instance with n variables and m clauses. Let $G = G(\sigma, q, \beta)$ and $\mathcal{P}$ be the graph and packing as constructed above and let $\overline{b} = |\mathcal{P}| + tp/(q+1)$. If σ has a satisfying truth assignment τ, then there is a solution φ of the* DELETION TO q-COLORABLE *instance $(G, \overline{b})$.*

Proof sketch. The proof is very similar to the proof of Theorem 13.3.6. Let τ_i be the partial truth assignment of variable group i induced by τ. We set $\varphi\big|_{U_i} = \kappa_i(\tau_i) \in \Phi_i$ for all i and $\varphi(f_s) = s$ for all $s \in [q]$. By definition of Φ_i, only the budget $|\mathcal{P}|$ for the packing $\mathcal{P}$ remains, hence on every graph in $\mathcal{P}$, we can perform exactly one deletion and nowhere else.

We propagate deletions along thin arrows and extend the solution across decoding gadgets $S_{\{\varphi_i(u)\}\setminus\{\perp\}}(\{u\}, v)$ as before. For every Y^j_{i,φ_i}, we delete the distinguished vertex $\hat{y}^j_{i,\varphi_i}$ and for every j-th clause, group i, and solution $\varphi_i \neq \psi_i \in \Phi_i$, we pick one of the vertices in the K_q of Y^j_{i,φ_i} and delete it. This concludes the description of the deletions.

It remains to show that the remaining graph can be properly q-colored. Comparing to the proof of Theorem 13.3.6, the arguments change slightly for $L_{i,S}$ and Y^j_{i,ψ_i}. Consider some copy of some $L_{i,S}$, due to $|S| + |\varphi^{-1}(\perp) \cap U_i| = (\frac{q}{q+1}p + 1) + p/(q+1) = p + 1 > |U_i|$ there is a least one $u \in S$ with $\varphi(u) = \perp$. Therefore, the thin arrow from u to $L_{i,S}$ propagates a deletion to $L_{i,S}$ and this $(q+1)$-critical graph is resolved.

Consider some Y^j_{i,ψ_i} with $\psi_i \in \Phi_i$. If $\psi_i \neq \varphi_i$, the argument works as in the proof of Theorem 13.3.6. If $\psi_i = \varphi_i$, then we claim that the large independent set of Y^j_{i,φ_i} is fully deleted. The large independent set has size $p + 1 = |U_i| + 1$. We distributed one deletion to $\hat{y}^j_{i,\varphi_i}$ and all other vertices v in the large independent set are hit by some $W^j_{i,\varphi_i,u}$, $u \in U_i$. Since $\varphi_i(u) \in \{\varphi_i(u)\}$, we see that $W^j_{i,\varphi_i,u} = S_{\{\varphi_i(u)\}\setminus\{\perp\}}(\{u\}, v)$ propagates a deletion to v due to Lemma 13.3.5. So, the large independent set is indeed fully deleted and thereby the complete $(q+1)$-partite graph is resolved. □

Theorem 13.4.5. *Let σ be a* SATISFIABILITY *instance with n variables and m clauses. Let $G = G(\sigma, q, \beta)$ and $\mathcal{P}$ be the graph and packing as constructed above and let $\overline{b} = |\mathcal{P}| + tp/(q+1)$. If φ is a solution of the* DELETION TO q-COLORABLE *instance $(G, \overline{b})$, then $|\varphi^{-1}(\perp)| = \overline{b}$. Furthermore, there is a normalized solution ψ with $|\psi^{-1}(\perp)| = |\varphi^{-1}(\perp)| = \overline{b}$ and $\psi\big|_{U_i} \in \Phi_i$ for all $i \in [t]$.*

Proof. Since the packing $\mathcal{P}$ consists only of $(q+1)$-critical graphs, at least $|\mathcal{P}|$ deletions must be performed by φ on these graphs. The remainder of the deletions is performed on the central vertices. Suppose that there is some group $i \in [t]$ such that $|\varphi^{-1}(\perp) \cap U_i| < p/(q+1)$, then there exists an $S \subseteq U_i$ with $|S| = \frac{q}{q+1}p + 1$ and $S \cap \varphi^{-1}(\perp) = \emptyset$. Consider some copy of the corresponding $L_{i,S}$ and notice that all thin arrows leading to $L_{i,S}$ are passive unless we pay for extra deletions. Since $L_{i,S}$ is an $(q+1)$-critical graph that does not belong to $\mathcal{P}$, we must perform one additional deletion per copy of $L_{i,S}$. There are $1 + tp/(q+1) > \overline{b} - |\mathcal{P}|$ copies of $L_{i,S}$, hence this would exceed the available budget. So, we can conclude that $|\varphi^{-1}(\perp) \cap U_i| \geq p/(q+1)$ for all groups $i \in [t]$.

Together with $|\varphi^{-1}(\perp)| \leq \overline{b} = |\mathcal{P}| + tp/(q+1)$, we see that $|\varphi^{-1}(\perp)| = \overline{b}$ and $|\varphi^{-1}(\perp) \cap U_i| = p/(q+1)$ for all $i \in [t]$. Hence, φ cannot delete any vertex of the central clique F and by permuting the colors, we obtain the desired normalized solution ψ with $\psi\big|_{U_i} \in \Phi_i$. □

Theorem 13.4.6. *Let σ be a* SATISFIABILITY *instance with n variables and m clauses. Let $G = G(\sigma, q, \beta)$ and $\mathcal{P}$ be the graph and packing as constructed above and let $\overline{b} = |\mathcal{P}| + tp/(q+1)$. If the* DELETION TO q-COLORABLE *instance $(G, \overline{b})$ has a solution φ, then σ has a satisfying truth assignment τ.*

Proof sketch. This proof is very similar to the proof of Theorem 13.3.8. We first invoke Theorem 13.4.5, which implies that $|\varphi^{-1}(\bot)| = \bar{b}$ and allows us to assume that φ is a normalized solution with $\varphi_i := \varphi|_{U_i} \in \Phi_i$. We only diverge from the proof of Theorem 13.3.8 when proving that $\varphi(\hat{y}^j_{i,\psi_i}) = \bot$ only if $\psi_i = \varphi_i$.

As before, $\hat{y}^j_{i,\psi_i}$ may only be deleted if the large independent set of Y^j_{i,ψ_i} is fully deleted. Now, consider some $\psi_i \in \Phi_i \setminus \{\varphi_i\}$. We will distinguish two cases.

1. Suppose that $\psi_i^{-1}(\bot) \neq \varphi_i^{-1}(\bot)$. Due to $\varphi_i, \psi_i \in \Phi_i$, there exists some $u \in U_i$ with $\psi_i(u) = \bot$ and $\varphi_i(u) \neq \bot$. The associated $W^j_{i,\psi_i,u} = S_\emptyset(\{u\}, v)$ cannot propagate a deletion to v in this case by Lemma 13.3.5 and hence the large independent set of Y^j_{i,ψ_i} is not fully deleted.
2. Suppose that $\psi_i^{-1}(\bot) = \varphi_i^{-1}(\bot)$, then $\psi_i \neq \varphi_i$ implies that there exists some $u \in U_i \setminus \psi_i^{-1}(\bot) = U_i \setminus \varphi_i^{-1}(\bot)$ with $\psi_i(u) \neq \varphi_i(u)$. Again, the associated $W^j_{i,\psi_i,u} = S_{\{\psi_i(u)\}}(\{u\}, v)$ cannot propagate a deletion to v in this case by Lemma 13.3.5 and, again, the large independent set of Y^j_{i,ψ_i} is not fully deleted.

This proves the claim regarding the deletion of $\hat{y}^j_{i,\psi_i}$. □

Theorem 13.4.7. *Let $G = G(\sigma, q, \beta)$ be the graph as constructed above. The set $M = \left(\bigcup_{i=1}^t U_i\right) \cup \{f_s : s \in [q]\}$ is a modulator for G of size $tp + q$ to treewidth q, i.e., $|M| = tp + q$ and $\operatorname{tw}(G - M) \leq q$.*

Proof sketch. The proof is very similar to the proof of Theorem 13.3.9. The notable difference is that all thick arrows are replaced by thin arrows, but this does not affect the treewidth. The connected components corresponding to a clause have for the sake of the omniscient cops-and-robber-game the same structure as before, only the number of $W^j_{i,\varphi_i,u}$ incident to some Y^j_{i,φ_i} changes. □

Proof of Theorem 13.4.1. Suppose there is some $q \geq 2$ and $\varepsilon > 0$ such that DELETION TO q-COLORABLE can be solved in time $\mathcal{O}^*((q+1-\varepsilon)^{|M|})$, where M is a modulator to treewidth q. We will show that there exists a $\delta < 1$ such that we can solve SATISFIABILITY in time $\mathcal{O}(2^{\delta n} \operatorname{poly}(m)) = \mathcal{O}^*(2^{\delta n})$ where n is the number of variables. This contradicts CNF-SETH and hence implies the desired lower bound.

Given a SATISFIABILITY instance σ, we construct the graph $G = G(\sigma, q, \beta)$, where β depends only on q and ε and will be chosen later. We can consider q and ε as constants, hence also $\beta = \beta(q, \varepsilon)$ and $p = p(\beta)$ are constant. As in the proof of Theorem 13.3.1, we can again see that G has size polynomial in n and m and can be constructed in polynomial time. Furthermore, we can also obtain a tree decomposition of $G' := G - M$ from the proof of Theorem 13.4.7 in polynomial time.

In the construction of $G(\sigma, q, \beta)$, we chose p as the smallest integer that is divisible by $(q+1)$ and satisfies $(q+1)^p \frac{p!q!}{(p+q)!} \geq 2^\beta$. We let $\gamma = \lceil \log_{q+1}(2^\beta) \rceil$ and show that $p \leq \gamma + 3q\lceil \log_{q+1} \gamma \rceil + (q+1)$. The summand $(q+1)$ will ensure divisibility. We compute that

$$\begin{aligned}(q+1)^p \frac{p!q!}{(p+q)!} \geq (q+1)^p (2p)^{-q} &= \frac{(q+1)^\gamma \gamma^{3q}}{2^q(\gamma + 3q\lceil \log_{q+1} \gamma \rceil + (q+1))^q} \geq \frac{(q+1)^\gamma \gamma^{3q}}{2^q (5q\gamma)^q} \\ &= (q+1)^\gamma \frac{\gamma^{2q}}{(10q)^q} \geq (q+1)^\gamma \geq 2^\beta,\end{aligned}$$

where we use throughout that β and hence γ is sufficiently large.

Running the assumed algorithm for DELETION TO q-COLORABLE on G decides, by Theorem 13.3.6 and Theorem 13.3.8, the satisfiability of σ. Since G can be constructed in polynomial time, we can solve SATISFIABILITY in time

$$\begin{aligned}\mathcal{O}^*((q+1-\varepsilon)^{|M|}) &= \mathcal{O}^*((q+1-\varepsilon)^{\lceil \frac{n}{\beta} \rceil p+q}) \leq \mathcal{O}^*((q+1-\varepsilon)^{\frac{n}{\beta}p}) \\ &\leq \mathcal{O}^*((q+1-\varepsilon)^{\frac{n}{\beta}\log_{q+1}(2^\beta)}(q+1-\varepsilon)^{\frac{n}{\beta}(3q\lceil \log_{q+1}(\gamma)\rceil+q+2)}).\end{aligned}$$

For the first factor, we see that

$$\mathcal{O}^*((q+1-\varepsilon)^{\frac{n}{\beta}\log_{q+1}(2^\beta)}) = \mathcal{O}^*((q+1-\varepsilon)^{n\log_{q+1}(2)}) \leq \mathcal{O}^*(2^{\delta_1 n}),$$

where $\delta_1 < 1$. We bound the exponent in the second factor as follows

$$\begin{aligned}\frac{n}{\beta}(3q\lceil \log_{q+1}(\gamma)\rceil + q + 2) &\leq \frac{n}{\beta}(6q\log_{q+1}(\gamma)) \leq \frac{n}{\beta}(6q\log_{q+1}(p_0^2)) \\ &= \frac{n}{\beta}(12q\log_{q+1}(\beta)) = 12qn\frac{\log_{q+1}(\beta)}{\beta},\end{aligned}$$

where we use $\gamma \leq p_0^2$ in the second inequality. This bound shows that

$$\mathcal{O}^*\left((q+1-\varepsilon)^{\frac{n}{\beta}(3q\lceil \log_{q+1}(\gamma)\rceil+q+2)}\right) \leq \mathcal{O}^*\left(\left((q+1-\varepsilon)^{12q\frac{\log_{q+1}(\beta)}{\beta}}\right)^n\right) \leq \mathcal{O}^*(2^{\delta_2 n}),$$

where we can choose δ_2 arbitrarily close to 0 by making β large enough.

By choosing δ_2 so that $\delta := \delta_1 + \delta_2 < 1$, we obtain that SATISFIABILITY can be solved in time $\mathcal{O}^*\left(2^{\delta n}\right)$, hence CNF-SETH, Conjecture 2.1.1 has to be false. □

Corollary 13.4.8. *If* DELETION TO q-COLORABLE *can be solved in time* $\mathcal{O}^*((q+1-\varepsilon)^{\mathrm{td}(G)})$ *for some* $q \geq 2$ *and* $\varepsilon > 0$*, then CNF-SETH is false.*

Proof. Follows from Theorem 13.4.1 and Corollary 2.4.18. □

14 Algorithm for Deletion to q-Colorable

In this section we describe how to solve DELETION TO q-COLORABLE in time $\mathcal{O}^*((2^q)^k)$ given a k-multi-expression μ for $G = (V, E)$, the case $q = 1$, i.e., VERTEX COVER, without costs was already proven by Fürer [79]. We assume that the given costs $\mathbf{c}(v)$, $v \in V$, are polynomially bounded in the number of vertices $n = |V|$ and we perform bottom-up dynamic programming along the associated syntax tree T_μ. We again view *solutions* to DELETION TO q-COLORABLE as functions $\varphi\colon V(G) \to [q] \cup \{\perp\}$ with the property discussed in the outline, cf. Section 13.1.

Basic Definitions for Dynamic Programming. We extend the definitions for dynamic programming on clique-expressions given in Section 5.1 to multi-clique-expressions in the natural way. So, for $t \in V(T_\mu)$, we denote by μ_t the subexpression induced by the subtree of T_μ rooted at t. For a fixed k-multi-clique-expression μ, we define $G_t = G_{\mu_t}$, $V_t = V(G_t)$, $E_t = E(G_t)$, and $\mathtt{slab}_t = \mathtt{slab}_{\mu_t}$ for any $v \in V(T_\mu)$. Furthermore, we write $V_t^\ell = \mathtt{slab}_t^{-1}(\{S \subseteq [k] : \ell \in S\})$ for the set of all vertices with label ℓ at node t.

Theorem 14.0.1. *Given a k-expression μ for $G = (V, E)$,* DELETION TO q-COLORABLE *on G can be solved in time $\mathcal{O}^*((2^q)^k)$.*

Proof. Let $(G, \bar{b})$ be a DELETION TO q-COLORABLE instance and μ a k-multi-expression for G. Similar to Courcelle and Olariu [44], we can assume by using local swapping transformations that between every two union-operations at most $\mathcal{O}(k^2)$ join-operations, and $\mathcal{O}(k)$ (set-)relabel- and delete-operations occur. Since μ contains $\mathcal{O}(n)$ introduce- and union-operations, this shows that T_μ contains at most $\mathcal{O}(k^2 n) = \mathcal{O}^*(1)$ nodes. We replace every set-relabel operation $\rho_{i\to S}$ with $|S| \geq 3$, $S = \{\ell_1, \ldots, \ell_s\}$, by a sequence of set-relabel operations $\rho_{i\to\{i,\ell_1\}}, \rho_{i\to\{i,\ell_2\}}, \ldots, \rho_{i\to\{i,\ell_{s-2}\}}, \rho_{i\to\{\ell_{s-1},\ell_s\}}$, thus we only need to handle set-relabels $\rho_{i\to S}$ with $|S| \leq 2$.

For every node $t \in V(T_\mu)$ and label ℓ, we store the set of colors used on V_t^ℓ. After deleting the appropriate vertices, the remaining graph should be q-colorable, hence the possible color sets are precisely the subsets of $[q]$, where $\emptyset$ indicates that all vertices are deleted. Since we use at most k labels at every node, this results in $(2^q)^k$ possible types of partial solutions at each node. If the work for each type is only polynomial, then the claimed running time immediately follows, since there are only a polynomial number of nodes in $V(T_\mu)$.

For every $t \in V(T_\mu)$, the family $\mathcal{A}_t$ of *partial solutions at* t consists of all $\varphi\colon V_t \to [q] \cup \{\perp\}$ such that φ induces a q-coloring of $G_t - \varphi^{-1}(\perp)$. We say that a function $f\colon [k] \to \mathcal{P}([q])$ is a *signature* and the set of *partial solutions at* t *compatible with* f is defined by

$$\mathcal{A}_t(f) = \{\varphi \in \mathcal{A}_t : \ \varphi(V_t^\ell) \setminus \{\perp\} = f(\ell) \text{ for all } \ell \in [k]\}.$$

We want to compute the quantity $A_t(f) = \min\{\mathbf{c}(\varphi^{-1}(\perp)) : \varphi \in \mathcal{A}_t(f)\}$, where $A_t(f) = \infty$ if $\mathcal{A}_t(f) = \emptyset$. Let t_0 be the root node of the k-expression μ. The algorithm returns true if there is a t_0-signature f such that $A_{t_0}(f) \leq \bar{b}$ and otherwise the algorithm returns false.

Note that $f(\ell) = \emptyset$ implies $\varphi(V_t^\ell) = \{\perp\}$ for all $\varphi \in \mathcal{A}_t(f)$, i.e., all vertices with label ℓ are deleted. Furthermore, the definition of $\mathcal{A}_t(f)$ implies that $\mathcal{A}_t(f) = \emptyset$ and $A_t(f) = \infty$ whenever $|f(\ell)| > |V_t^\ell|$ for some $\ell \in [k]$; we will not explicitly mention this edge case again in what follows and assume that the considered f satisfy $|f(\ell)| \leq |V_t^\ell|$ for all $\ell \in [k]$. We proceed by presenting the recurrences to compute $A_t(f)$ for all nodes t and signatures f and afterwards show the correctness of these recurrences.

Base Case. If $t = \ell(v)$ for some $\ell \in [k]$, then $A_t(f) = [f(\ell) = \emptyset]\mathbf{c}(v)$, because the solution cost is $\mathbf{c}(v)$ if v is deleted and 0 otherwise.

Delete Case. If $t = \epsilon_i(G_{t'})$ for some $i \in [k]$, where t' is the child of t, then we simply need to try all possible states, i.e., subsets of $[q]$, for label i in the previous table and store the best result among them, resulting in the recurrence

$$A_t(f) = \min\{A_{t'}(f') : f'(\ell) = f(\ell) \text{ for all } \ell \in [k] \setminus \{i\}\}.$$

Note that the set on the right-hand side consists of at most $2^q = \mathcal{O}(1)$ numbers, hence the recurrence can be computed in polynomial time.

Simple Relabel Case. If $t = \rho_{i \to j}(G_{t'})$ for some $i \neq j \in [k]$ and where t' is the child of t, then the recurrence is given by

$$A_t(f) = \min\{A_{t'}(f') : f'(\ell) = f(\ell) \text{ for all } \ell \in [k] \setminus \{i,j\} \text{ and } f'(i) \cup f'(j) = f(j)\}.$$

By assumption, f will always satisfy $f(i) = \emptyset$ here, since there are no vertices with label i in G_t. This recurrence goes over all ways how the colors $f(j)$ used for vertices with label j in G_t can be split among the vertices with label i and j in the previous graph $G_{t'}$. Observe that we are taking the minimum over at most $(2^q)^2 = \mathcal{O}(1)$ numbers on the right-hand side, hence this recurrence can be computed in polynomial time.

Set Relabel Case. Due to the presented transformation, we only have to consider set-relabels with $t = \rho_{i \to S}(G_{t'})$, where $|S| \leq 2$ and t' is the child of t. The delete case and simple relabel case already take care of $|S| \leq 1$, so it remains to consider $|S| = 2$. Let $S = \{j_1, j_2\}$. Similar to the simple relabel case, we have to consider all ways the colors $f(j_1)$

are split among vertices with label i and j_1 at the previous node and analogously for $f(j_2)$. So, the recurrence is given by

$$A_t(f) = \min\{A_{t'}(f') : f'(\ell) = f(\ell) \text{ for all } \ell \in [k] \setminus \{i, j_1, j_2\}, \\ f'(i) \cup f'(j_1) = f(j_1) \text{ and } f'(i) \cup f'(j_2) = f(j_2)\}.$$

On the right-hand side, we are taking the minimum over at most $(2^q)^3 = \mathcal{O}(1)$ numbers, hence the recurrence can be computed in polynomial time.

Join Case. If $t = \eta_{i,j}(G_{t'})$ for some $i \neq j \in [k]$, where t' is the child of t, and assuming without loss of generality that $V_{t'}^i \neq \emptyset$ and $V_{t'}^j \neq \emptyset$, then

$$A_t(f) = \begin{cases} A_{t'}(f) & \text{if } f(i) \cap f(j) = \emptyset, \\ \infty & \text{else.} \end{cases}$$

This recurrence filters out all partial solutions where the coloring properties are not satisfied at some newly added edge. This happens precisely when $f(i) \cap f(j) \neq \emptyset$, because then there exists an edge in the join between label i and j whose endpoints get the same color.

Union Case. If $t = G_{t_1} \oplus G_{t_2}$ where t_1 and t_2 are the children of t, then

$$A_t(f) = \min\{A_{t_1}(f_1) + A_{t_2}(f_2) : f_1(\ell) \cup f_2(\ell) = f(\ell) \text{ for all } \ell \in [k]\}.$$

Here, we assume that $\infty + x = x + \infty = \infty + \infty = \infty$ for all $x \in \mathbb{N}$. This recurrence goes for each label $\ell \in [k]$ over all ways how the color set $f(\ell)$ can be split among the vertices with label ℓ in the first graph G_{t_1} and in the second graph G_{t_2}.

To compute this recurrence, we make use of the following fast cover product algorithm.

Lemma 14.0.2 ([47]). *For two functions $f, g \colon \mathcal{P}(U) \to \{-M, \ldots, M\}$, given all $2^{|U|}$ values of f and g in the input, all the $2^{|U|}$ values of the* cover product $f *_c g$ *defined by*

$$(f *_c g)(X) = \min_{Y \cup Z = X} f(Y) + g(Z),$$

can be computed in time $2^{|U|}|U|^{\mathcal{O}(1)} \cdot \mathcal{O}(|M| \log |M| \log\log |M|)$.

Lemma 14.0.2 allows us to compute the recurrence for all f simultaneously in time $\mathcal{O}^*((2^q)^k)$: We interpret the signatures $f \colon [k] \to \mathcal{P}([q])$ as subsets of $[k] \times [q]$ in the following way: $S(f) = \{(i, c) : i \in [k], c \in f(i)\}$. Observe that $f_1(\ell) \cup f_2(\ell) = f(\ell)$ for all $\ell \in [k]$ is equivalent to $S(f_1) \cup S(f_2) = S(f)$. Now, A_t can be considered as a function $\mathcal{P}([k] \times [q]) \to [\mathbf{c}(V) + 1]$ by replacing ∞ with $\mathbf{c}(V) + 1$, since we have $A_t(f) \leq \mathbf{c}(V)$ whenever $\mathcal{A}_t(f) \neq \emptyset$. The recurrence of the union case can then be considered as the $(\min, +)$-cover product of A_{t_1} and A_{t_2}. By Lemma 14.0.2 with $U = [k] \times [q]$ and $M = \mathbf{c}(V) + 1$, we can compute all values of A_t in time $2^{kq}(kq)^{\mathcal{O}(1)} \cdot n^{\mathcal{O}(1)} = \mathcal{O}^*((2^q)^k)$, since $\mathbf{c}(V)$ is polynomial in n by assumption.

Correctness. We prove the correctness by bottom-up induction along the syntax tree T_μ. In the base case, G_t only consists of the single vertex v and we can either delete v or assign some color to v. Together with the edge case handling, this is implemented by the formula for the base case.

For the delete case, notice that $\mathcal{A}_t = \mathcal{A}_{t'}$, $V_t^i = \emptyset$, and $V_t^\ell = V_{t'}^\ell$ for all $\ell \in [k] \setminus \{i\}$. If $\varphi \in \mathcal{A}_t(f)$, then $\varphi \in \mathcal{A}_{t'}(f')$, where $f' = f[i \mapsto (\varphi(V_{t'}^i) \setminus \{\bot\})]$. In the other direction, if $\varphi \in \mathcal{A}_{t'}(f')$, then $\varphi \in \mathcal{A}_t(f)$, where $f = f'[i \mapsto \emptyset]$. Therefore, the correctness of the recurrence for the delete case follows.

For the simple relabel case, notice that $G_t = G_{t'}$, $V_t^i = \emptyset$, $V_t^j = V_{t'}^i \cup V_{t'}^j$, and $V_t^\ell = V_{t'}^\ell$ for all $\ell \in [k] \setminus \{i, j\}$. Let f' be a candidate in the recurrence of $A_t(f)$ and $\varphi' \in \mathcal{A}_{t'}(f')$ be a minimizer in the definition of $A_{t'}(f')$, then we also have that $\varphi' \in \mathcal{A}_t(f)$ since $\varphi'(V_t^j) \setminus \{\bot\} = (\varphi'(V_{t'}^i) \setminus \{\bot\}) \cup (\varphi'(V_{t'}^j) \setminus \{\bot\}) = f'(i) \cup f'(j) = f(j)$. Hence, the recurrence is an upper bound on $A_t(f)$.

In the other direction, let φ be a minimizer in the definition of $A_t(f)$ and consider f' with $f'(\ell) = \varphi(V_{t'}^\ell) \setminus \{\bot\}$ for all $\ell \in [k]$. Then f' satisfies $f'(i) \cup f'(j) = f(j)$ and $\varphi \in \mathcal{A}_{t'}(f')$, so f' is also considered in the recurrence and the recurrence is a lower bound on $A_t(f)$.

For the set relabel case, we have the equations $G_t = G_{t'}$, $V_t^{j_1} = V_{t'}^i \cup V_{t'}^{j_1}$, $V_t^{j_2} = V_{t'}^i \cup V_{t'}^{j_2}$, and $V_t^\ell = V_{t'}^\ell$ for all $\ell \in [k] \setminus \{i, j_1, j_2\}$, thus allowing for a very similar proof to the simple relabel case, which we omit here.

For the join case, notice that for $\varphi \in \mathcal{A}_{t'}(f) \supseteq \mathcal{A}_t(f)$ it holds that $\varphi \in \mathcal{A}_t(f)$ if and only if $\varphi(V_{t'}^i) \cap \varphi(V_{t'}^j) \subseteq \{\bot\}$ as otherwise φ cannot induce a coloring of $G_t - \varphi^{-1}(\bot)$.

For the union case, a feasible solution φ of G_t induces feasible solutions $\varphi_1 = \varphi\big|_{V_{t_1}}$ of G_{t_1} and $\varphi_2 = \varphi\big|_{V_{t_2}}$ of G_{t_2} such that $\varphi_1(V_t^\ell) \cup \varphi_2(V_t^\ell) = \varphi(V_t^\ell)$ for all $\ell \in [k]$ and vice versa. □

Together with the lower bound for DELETION TO q-COLORABLE[$\mathrm{tcm}_{\mathrm{tw} \leq q}$] shown in Chapter 13 and the parameter relationships in Section 2.4.4, this shows that the base 2^q is optimal for all parameters sandwiched between $\mathrm{tcm}_{\mathrm{tw} \leq q}$ and mcw, which in particular include clique-width, modular-treewidth, and twinclass-treedepth.

15 Conclusion and Future Work

We have studied in this part how the fine-grained time complexity and space complexity of graph problems is affected by going beyond width parameters to the more restrictive depth parameters and modulators. First, we saw how the cut-and-count-technique is the right tool to obtain polynomial-space branching algorithms for connectivity problems parameterized by treedepth, while simultaneously maintaining the running time base of the treewidth algorithms. While we have not given the proofs, our techniques also extend to other problems like CONNECTED FEEDBACK VERTEX SET and CONNECTED TOTAL DOMINATING SET.

Secondly, we investigated the fine-grained time complexity of several vertex-deletion problems relative to depth-parameterizations and modulator-parameterizations. By giving novel lower bound constructions, we have established for several natural problems that a small modulator to a simple graph class is already as hard as small treewidth. In the dense setting, something even stronger occurs: A modulator consisting of a few twinclasses is sufficient to obtain the tight lower bound, as opposed to several disjoint separators or more involved dense structure than what is captured by twinclasses.

We survey some possible research directions and future work regarding depth parameters and modulators in this section and additional research topics are discussed in Chapter 17.

Faster Algorithms for Connectivity Problems. We lack tight lower bounds for any of the connectivity problems parameterized by treedepth. Hence, it is a natural question whether the running times that we have obtained via the cut-and-count-technique can be beaten, even faster algorithms using exponential space would be interesting, e.g., can STEINER TREE[treedepth] be solved in time $\mathcal{O}^*((3-\varepsilon)^k)$ for some $\varepsilon > 0$? This question extends to the edge-based connectivity problems considered by Nederlof et al. [137]. A notable example is HAMILTONIAN CYCLE[treedepth], which is solvable in time $\mathcal{O}^*((2+\sqrt{2})^k)$ due to the corresponding time-optimal algorithm for HAMILTONIAN CYCLE[pathwidth] by Cygan, Kratsch, and Nederlof [48], but it is open whether this time is also optimal for treedepth. The polynomial-space algorithm obtained by Nederlof et al. [137] for HAMILTONIAN CYCLE[treedepth] has the larger running time $\mathcal{O}^*(5^k)$; can this running time be improved while maintaining polynomial space?

Derandomization. As observed by Nederlof et al. [137], the techniques to obtain deterministic algorithms for connectivity problems parameterized by width parameters, i.e., the rank-based approach and squared determinant approach, see Section 3.1.3, do not seem to transfer to the polynomial-space treedepth-setting. The rank-based approach stores solution sets which can have an exponential size and the squared determinant approach leads to complicated convolutions, where it is unclear how to bring them into the appropriate form such that they allow for polynomial-space branching algorithms. Consequently, obtaining

such deterministic algorithms in this context seems to require novel tools and would be of great interest. In a similar vein, the inability to apply these techniques also means that we do not know how to solve counting or (exponentially-)weighted variants of connectivity problems parameterized by treedepth in single-exponential time and polynomial space.

Dominating Set Parameterized by Treedepth. While we have somewhat remedied the situation by providing tight lower bounds for VERTEX COVER, ODD CYCLE TRANSVERSAL, and MAXIMUM CUT parameterized by treedepth, the lack of tight lower bounds for treedepth-parameterization does extend beyond connectivity problems. A natural benchmark problem for which we lack such a lower bound is DOMINATING SET[treedepth], i.e., can DOMINATING SET[treedepth] be solved in time $\mathcal{O}^*((3-\varepsilon)^k)$ for some $\varepsilon > 0$; the same question was also asked by Chen et al. [37]. Chen et al. also show that single-pass dynamic programming algorithms are not able to achieve such an improvement. A folklore algorithm obtains running time $\mathcal{O}^*(2^k)$ for DOMINATING SET[vertex cover], so we do know that modulator-parameterizations can allow for improved algorithms. What about DOMINATING SET[feedback vertex set] or DOMINATING SET[modulator to treewidth 2]? We remark that the $\mathcal{O}^*(3^k)$-time and polynomial-space algorithm for DOMINATING SET[treedepth] of Pilipczuk and Wrochna [155] also counts the number of dominating sets; can one show that this algorithm is optimal for the counting variant of DOMINATING SET[treedepth]?

Approximating Treedepth. When a treedepth decomposition is not given by other means, then applying the presented algorithms requires that we compute some treedepth decomposition first. Czerwiński et al. [52] provide an $\mathcal{O}(\mathrm{td}(G)\log^{3/2}\mathrm{td}(G))$-approximation for treedepth in polynomial time and Nadara et al. [135] show that treedepth can be computed exactly in time $2^{\mathcal{O}(k^2)}n$ and polynomial space. However, composing either of those two algorithms with our single-exponential branching algorithms yields a running time that is not single-exponential anymore. Hence, it has been repeatedly asked, e.g. by Nederlof et al. [137] and Czerwiński et al. [52], whether a constant-factor approximation of treedepth is possible in single-exponential time and, ideally, in polynomial space.

Meta-Theorem for Saving Space Parameterized by Treedepth. Since it can be nontrivial to obtain polynomial-space algorithms with single-exponential running times parameterized by treedepth, an interesting question is which problem classes can already be handled by our current techniques. A starting point for a logic-based meta-theorem could be the results of Pilipczuk [152], who has presented a logic capturing many of the problems that are solvable in single-exponential time when parameterizing by treewidth.

Further Exploration of Depth Parameters. Compared to width parameters, the space of depth parameters is still quite unexplored. Several depth parameters have been put forward to also capture dense graphs, namely *shrub-depth* and *sc-depth* [87, 88], and rank-depth [53]; they are all equivalent in the sense that a graph class $\mathcal{F}$ is bounded in any single one of these depth parameters if and only if $\mathcal{F}$ is bounded in all of them [53, 88]. Shrub-depth is the most well-studied among these three depth parameters [38, 53, 57, 81, 82, 87, 88, 121, 153], with a focus on structural graph theory and finite model theory. However, shrub-depth

has the drawback that any finite graph has shrub-depth at most one, indeed it is more useful as an asymptotic concept distinguishing between graph classes of bounded and unbounded shrub-depth. Little is known about the fine-grained parameterized complexity with respect to these dense depth parameters. Do they allow for similar space savings while preserving the time complexity of their width analogs? Is there a natural dense depth parameter that nicely fits into the fine-grained parameter hierarchy presented in Section 2.4.4?

Modular-Treedepth. As done for pathwidth and treewidth, we can also bring treedepth into the dense setting by considering its modular-variant. While modular-treedepth is a somewhat restricted generalization of treedepth, it is a natural question whether the running times obtained for connectivity problems parameterized by modular-treewidth in Chapter 6 can also be achieved by polynomial-space branching algorithms with respect to modular-treedepth. In regards to this, we remark that the reductions for STEINER TREE and CONNECTED DOMINATING SET also reduce the case of modular-treedepth to treedepth and can simply be pipelined with the polynomial-space branching algorithms presented in this part. For CONNECTED VERTEX COVER[modular-treedepth], the recurrences in the modular-treewidth-algorithm are simple enough so that such a transfer should be possible, but more needs to be done for FEEDBACK VERTEX SET[modular-treedepth].

Modulators to Other Graph Classes. We have proven tight lower bounds for VERTEX COVER[$\mathrm{dist}_{\mathrm{pw}\leq 2}$] and DELETION TO q-COLORABLE[$\mathrm{dist}_{\mathrm{tw}\leq q}$], as mentioned in Section 10.2 the pathwidth bound for VERTEX COVER and the treewidth bound for $q = 2$, i.e., ODD CYCLE TRANSVERSAL, cannot be improved. Does DELETION TO q-COLORABLE[$\mathrm{dist}_{\mathrm{tw}\leq q-1}$] also admit an improved algorithm for $q > 2$? Jaffke and Jansen [109] give a dichotomy when q-COLORING[$\mathrm{dist}_{\mathcal{F}}$] admits improved algorithms for families $\mathcal{F}$ that exclude some biclique and are closed under induced subgraphs. Can similar dichotomies be obtained for VERTEX COVER or ODD CYCLE TRANSVERSAL? For minor-closed families $\mathcal{F}$, Bougeret et al. [28] give a dichotomy when VERTEX COVER[$\mathrm{dist}_{\mathcal{F}}$] admits a polynomial kernel; how is this result related to a dichotomy about improved algorithms? How do these dichotomies change when considering twinclass-modulators or some other notion of dense modulators instead?

Notions of Dense Modulators. Even more so than for depth parameters, there have been very few studies on which notions of dense modulators are fruitful. Our study of twinclass-modulators is in service of capturing very strong structural restrictions that still allow us to prove tight fine-grained lower bounds and is not focused on their general applicability of the concept. Any notion of dense modulators has to consider what structure is allowed inside a modulator and how a modulator is allowed to connect to the remainder of the graph. Eiben, Ganian, and Szeider [67, 66] study the concept of *well-structured modulators*, which are modulators that can be partitioned into few parts, where each part has small rank-width and induces a *split-module* of the graph; compared to modules, split-modules are more general as they can always contain vertices that have no neighbors outside the split-module. We pose as a concrete question if the $\mathcal{O}^*((2^q-2)^k)$ lower bound for q-COLORING[linear-clique-width] by Lampis [123] can be improved to hold already relative to some dense modulator;

twinclass-modulators to constant treewidth are not the appropriate concept, as the algorithm for q-Coloring[twinclass-treewidth] by Lampis [123] implies an improved running time in that case. A similar question can be asked about the fine-grained results of Ganian et al. [86] for the more general homomorphism problems parameterized by clique-width.

Part IV

Conclusion

16 Summary

In this thesis, we have studied the fine-grained parameterized complexity of various benchmark graph problems relative to structural parameters to obtain a precise understanding of how input structure affects problem complexity. Throughout this thesis, we have seen how this research is closely tied to the possible problem states at various kinds of separators. The main class of problems we consider are connectivity problems, whose complexity is harder to pin down due to the presence of a global connectivity constraint. Motivated by the success of the cut-and-count-technique by Cygan et al. [51] for obtaining tight algorithms for connectivity problems parameterized by treewidth and pathwidth, we applied the cut-and-count-technique in the setting of several other structural parameters.

Beginning with the dense width parameters *clique-width* and *modular-treewidth*, we obtain multiple tight results for connectivity problems relative to these parameters, where we complement the obtained cut-and-count-algorithms with lower bounds following the construction principle of Lokshtanov et al. [126]. Via a quite generic application of cut-and-count, we obtain many single-exponential algorithms relative to clique-width. Using problem-specific insights, we can obtain the optimal running time base for Connected Vertex Cover[clique-width] and Connected Dominating Set[clique-width]. However, as discussed in Chapter 9, for other connectivity problems parameterized by clique-width, optimal bases seem out of reach for our techniques.

For modular-treewidth, we observe that overarching savings are possible in the cut-and-count-technique compared to clique-width, as it suffices to only consider homogeneous cuts, see Lemma 6.1.4. In the cases of Steiner Tree and Connected Dominating Set, these savings even allow for a direct reduction to the treewidth-case, thus the more expressive modular-treewidth can be considered instead of treewidth with no loss in complexity. As the treewidth-techniques transfer more directly to modular-treewidth, we are able to obtain further optimal bases for Connected Vertex Cover[modular-treewidth] and Feedback Vertex Set[modular-treewidth]. However, the restrictiveness of modular-treewidth results in more challenging accompanying lower bound constructions compared to clique-width.

Going beyond width parameters to depth parameters, we apply the cut-and-count-technique in the context of treedepth. We show that the depth restriction allows us in several cases to transform the dynamic programming algorithms for treewidth into polynomial-space branching algorithms relative to treedepth without worsening the running time base. As previously stated in Chapter 10 and Chapter 15, the lower bound principle of Lokshtanov et al. [126] generally relies on constructions of arbitrarily large depth, thus making lower bounds relative to treedepth more difficult to obtain. Since additionally the connectivity constraint is difficult to handle, we are not able to complement these branching algorithms with tight lower bounds relative to treedepth.

Seeking to remedy this situation, we develop novel lower bound constructions relative to modulators that transfer to treedepth and its twinclass-variant. As coloring constraints are more well-behaved than a connectivity constraint or even a domination constraint, the obtained tight lower bounds apply to the problem DELETION TO q-COLORABLE, which generalizes VERTEX COVER and ODD CYCLE TRANSVERSAL. Structurally, this shows that for these problems already a single complex separator is difficult to handle, instead of needing a large number of disjoint separators as in the lower bounds relative to width parameters.

17 Future Work

We conclude by listing some further possible directions for future work that are not covered in Chapter 9 or Chapter 15.

Fine-Grained Meta-Theorems. To obtain a broader understanding of the precise impact of input structure on problem complexity, meta-theorems determining the optimal bases for a larger problem class are particularly useful. Most of the known fine-grained meta-theorems focus on (a variant of) homomorphism problems relative to sparse parameters [63, 72, 145, 146, 151]. Exceptions to this are the meta-theorem of Ganian et al. [86] for homomorphism problems parameterized by clique-width; Esmer et al. [68] give a meta-theorem for homomorphisms problems where edge or vertex deletions are allowed, thus generalizing DELETION TO q-COLORABLE, parameterized by treewidth; Focke et al. [71] present a meta-theorem for counting variants of generalized domination problems parameterized by treewidth; finally, Marx et al. [132] consider general factor problems parameterized by treewidth. Further meta-theorems relative to dense parameters would be interesting, e.g., for the domination problems considered by Focke et al. [71]. Moreover, there is currently no fine-grained meta-theorem involving connectivity problems.

More Fine-Grained Equivalences. Cygan et al. [46] show for several problems, such as q-HITTING SET or q-SET SPLITTING parameterized by the size of the ground set, that improving upon the base 2 in the running time is equivalent to falsifying SETH, see also Theorem 2.1.2. More directly relevant to our investigations are the results of Iwata and Yoshida [106], who show for any base $\alpha > 1$ that the following statements are equivalent:

- SATISFIABILITY[primal-treewidth] can be solved in time $\mathcal{O}^*(\alpha^k)$,
- INDEPENDENT SET[treewidth] can be solved in time $\mathcal{O}^*(\alpha^k)$,
- INDEPENDENT SET[clique-width] can be solved in time $\mathcal{O}^*(\alpha^k)$.

The results of Iwata and Yoshida have several nice features. First, their lower bounds hold under the assumption that SATISFIABILITY[primal-treewidth] cannot be solved in time $\mathcal{O}((2-\varepsilon)^k)$ for any $\varepsilon > 0$, which is a weaker assumption than SETH. Secondly, such equivalences show that if SETH or the weaker assumption turn out to be false, then we also obtain faster algorithms for the considered target problems, which is not true for our one-way reductions. Finally, these equivalences also rule out that, say, only one of the considered target problems has a faster algorithm. Can more of such equivalences be obtained? In particular, what about equivalences for problems where the optimal base is conjectured to be larger than 2?

Further Width Parameters. Even aside from connectivity problems, little is known about the (very) fine-grained complexity relative to width parameters besides cutwidth,

path/treewidth, clique-width, or their twinclass/modular-variants. One could consider boolean-width, which is smaller than clique-width, where Bui-Xuan et al. [30] show that INDEPENDENT SET[boolean-width] can be solved in time $\mathcal{O}^*(4^k)$ and DOMINATING SET[boolean-width] in time $\mathcal{O}^*(8^k)$; are these bases also optimal under SETH? For other dense width parameters, we often do not have running times of the form $\mathcal{O}^*(\alpha^k)$, e.g. INDEPENDENT SET[rank-width] can be solved in time $2^{\mathcal{O}(k^2)}n^{\mathcal{O}(1)}$ due to Bui-Xuan et al. [32] and Bergougnoux et al. [14] show that the quadratic dependence in the exponent cannot be avoided, unless ETH fails. For mim-width, essentially all known algorithms have running time $n^{\mathcal{O}(k)}$, see Bergougnoux et al. [10]. Repeated from Chapter 9, can we more precisely understand the running time exponent for specific problems parameterized by rank-width or mim-width? Finally, for twin-width, Bonnet et al. [25] show that INDEPENDENT SET can be solved in time $\mathcal{O}^*(k^2d^{2k})$, where k is the solution size and d the twin-width, and DOMINATING SET in time $\mathcal{O}^*(2^{2(d^2+1)(2+\log d)k})$; how much more can these running times be optimized and what about other problems, e.g., ODD CYCLE TRANSVERSAL?

Heterogeneous Parameterizations. The parameters considered in this thesis all arise from *homogeneous* decomposition schemes, i.e., every separator is measured according to the *same* criterion. The recently evolving study of *heterogeneous* or *hybrid parameterizations*, see also Section 10.2, brings forth parameters such as $\mathcal{H}$-treewidth and *$\mathcal{H}$-elimination distance*, which essentially extend the base cases of treewidth and treedepth, respectively, to allow for any graph from the family $\mathcal{H}$. However, not much is known about fine-grained parameterized complexity relative to such parameters. Jansen et al. [110] show that VERTEX COVER can be solved in time $\mathcal{O}^*(2^k)$ and ODD CYCLE TRANSVERSAL in time $\mathcal{O}^*(3^k)$ when parameterized by bipartite-treewidth. Eiben et al. [65] show that MAXIMUM CUT[$\mathcal{R}_w$-treewidth] can be solved in time $\mathcal{O}^*(2^k)$, where $\mathcal{R}_w$ denotes the graphs of *rank-width* at most w. What results can be obtained for similar extensions of, say, clique-width? What about heterogeneous width parameters that do not only extend the base cases, but allow to measure separators according to two incomparable criteria?

Multi-Clique-Width. Fürer [79] has introduced multi-clique-width and shown that INDEPENDENT SET, and hence also VERTEX COVER, can be solved in time $\mathcal{O}^*(2^k)$ parameterized by multi-clique-width, thus extending the previously known algorithms for clique-width without increasing the base. In Chapter 14, we see that the same can be done for the more general DELETION TO q-COLORABLE. Can the same be done for the algorithms presented in Part II, which in particular assume an irredundant clique-expression? Is there a natural problem with a single-exponential algorithm relative to clique-width where going to multi-clique-width increases the base? Since we have that $\mathrm{mcw} \preceq \text{mod-tw} \preceq \mathrm{tw}$ and $\mathrm{mcw} \preceq \mathrm{cw}$, see Section 2.4.4, algorithms relative to multi-clique-width can unify separate algorithms for (modular-)treewidth and clique-width with the same base, e.g. VERTEX COVER is a problem where this applies.

Computing (Multi-)Clique-Width. A big caveat in applying algorithms parameterized by clique-width is that we are lacking good algorithms for computing clique-expressions. The currently best algorithms rely on approximating clique-width via rank-width, see Oum

and Seymour [149] for the first such algorithm and Fomin and Korhonen [74] for the most recent one. However, the approximation via rank-width introduces an exponential error, therefore all single-exponential algorithms parameterized by clique-width become double-exponential algorithms unless we are given a clique-expression by other means. A first step towards better approximation algorithms for clique-width could be a fixed-parameter tractable algorithm with subexponential error. The same issues apply even more to the more expressive multi-clique-width, where the rank-width approximation leads to a double-exponential error.

Bibliography

[1] Akanksha Agrawal and M. S. Ramanujan. “Distance from Triviality 2.0: Hybrid Parameterizations”. In: *Combinatorial Algorithms - 33rd International Workshop, IWOCA 2022, Trier, Germany, June 7-9, 2022, Proceedings*. Ed. by Cristina Bazgan and Henning Fernau. Vol. 13270. Lecture Notes in Computer Science. Springer, 2022, pp. 3–20.

[2] Jochen Alber and Rolf Niedermeier. “Improved Tree Decomposition Based Algorithms for Domination-like Problems”. In: *LATIN 2002: Theoretical Informatics, 5th Latin American Symposium, Cancun, Mexico, April 3-6, 2002, Proceedings*. Ed. by Sergio Rajsbaum. Vol. 2286. Lecture Notes in Computer Science. Springer, 2002, pp. 613–628.

[3] Josh Alman and Virginia Vassilevska Williams. “A Refined Laser Method and Faster Matrix Multiplication”. In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*. Ed. by Dániel Marx. SIAM, 2021, pp. 522–539.

[4] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. “Complexity of Finding Embeddings in a k-Tree”. In: *SIAM Journal on Algebraic Discrete Methods* 8.2 (1987), pp. 277–284. eprint: `https://doi.org/10.1137/0608024`.

[5] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[6] Brage I. K. Bakkane and Lars Jaffke. “On the Hardness of Generalized Domination Problems Parameterized by Mim-Width”. In: *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*. Ed. by Holger Dell and Jesper Nederlof. Vol. 249. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 3:1–3:19.

[7] Eric T. Bax. “Inclusion and Exclusion Algorithm for the Hamiltonian Path Problem”. In: *Inf. Process. Lett.* 47.4 (1993), pp. 203–207.

[8] Mahdi Belbasi and Martin Fürer. “A Space-Efficient Parameterized Algorithm for the Hamiltonian Cycle Problem by Dynamic Algebraization”. In: *Computer Science - Theory and Applications - 14th International Computer Science Symposium in Russia, CSR 2019, Novosibirsk, Russia, July 1-5, 2019, Proceedings*. Ed. by René van Bevern and Gregory Kucherov. Vol. 11532. Lecture Notes in Computer Science. Springer, 2019, pp. 38–49.

[9] Benjamin Bergougnoux. “Matrix decompositions and algorithmic applications to (hyper)graphs.” PhD thesis. University of Clermont Auvergne, Clermont-Ferrand, France, 2019.

[10] Benjamin Bergougnoux, Jan Dreier, and Lars Jaffke. “A logic-based algorithmic meta-theorem for mim-width”. In: *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 3282–3304. eprint: `https://epubs.siam.org/doi/pdf/10.1137/1.9781611977554.ch125`.

[11] Benjamin Bergougnoux and Mamadou Moustapha Kanté. “Fast exact algorithms for some connectivity problems parameterized by clique-width”. In: *Theor. Comput. Sci.* 782 (2019), pp. 30–53.

[12] Benjamin Bergougnoux and Mamadou Moustapha Kanté. “More Applications of the d-Neighbor Equivalence: Acyclicity and Connectivity Constraints”. In: *SIAM J. Discret. Math.* 35.3 (2021), pp. 1881–1926.

[13] Benjamin Bergougnoux, Mamadou Moustapha Kanté, and O-joung Kwon. “An Optimal XP Algorithm for Hamiltonian Cycle on Graphs of Bounded Clique-Width”. In: *Algorithmica* 82.6 (2020), pp. 1654–1674.

[14] Benjamin Bergougnoux, Tuukka Korhonen, and Jesper Nederlof. “Tight Lower Bounds for Problems Parameterized by Rank-Width”. In: *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*. Ed. by Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté. Vol. 254. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 11:1–11:17.

[15] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. “Trimmed Moebius Inversion and Graphs of Bounded Degree”. In: *Theory Comput. Syst.* 47.3 (2010), pp. 637–654.

[16] Andreas Björklund, Thore Husfeldt, Petteri Kaski, et al. “Fast Zeta Transforms for Lattices with Few Irreducibles”. In: *ACM Trans. Algorithms* 12.1 (2016), 4:1–4:19.

[17] Hans L. Bodlaender. “A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth”. In: *SIAM J. Comput.* 25.6 (1996), pp. 1305–1317.

[18] Hans L. Bodlaender. “A Partial k-Arboretum of Graphs with Bounded Treewidth”. In: *Theor. Comput. Sci.* 209.1-2 (1998), pp. 1–45.

[19] Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. “Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth”. In: *Inf. Comput.* 243 (2015), pp. 86–111.

[20] Hans L. Bodlaender, Jitender S. Deogun, Klaus Jansen, et al. “Rankings of Graphs”. In: *SIAM J. Discret. Math.* 11.1 (1998), pp. 168–181.

[21] Hans L. Bodlaender, John R. Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. “Approximating Treewidth, Pathwidth, Frontsize, and Shortest Elimination Tree”. In: *J. Algorithms* 18.2 (1995), pp. 238–255.

[22] Hans L. Bodlaender and Klaus Jansen. “On the Complexity of the Maximum Cut Problem”. In: *Nord. J. Comput.* 7.1 (2000), pp. 14–31.

[23] Hans L. Bodlaender, Erik Jan van Leeuwen, Johan M. M. van Rooij, and Martin Vatshelle. “Faster Algorithms on Branch and Clique Decompositions”. In: *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*. Ed. by Petr Hlinený and Antonín Kucera. Vol. 6281. Lecture Notes in Computer Science. Springer, 2010, pp. 174–185.

[24] Narek Bojikian, Vera Chekan, Falko Hegerfeld, and Stefan Kratsch. “Tight Bounds for Connectivity Problems Parameterized by Cutwidth”. In: *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, Hamburg, Germany, March 7-9, 2023*. To appear.

[25] Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. “Twin-width III: Max Independent Set, Min Dominating Set, and Coloring”. In: *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*. Ed. by Nikhil Bansal, Emanuela Merelli, and James Worrell. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 35:1–35:20.

[26] Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. “Twin-width I: Tractable FO Model Checking”. In: *J. ACM* 69.1 (2022), 3:1–3:46.

[27] Glencora Borradaile and Hung Le. “Optimal Dynamic Program for r-Domination Problems over Tree Decompositions”. In: *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*. Ed. by Jiong Guo and Danny Hermelin. Vol. 63. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 8:1–8:23.

[28] Marin Bougeret, Bart M. P. Jansen, and Ignasi Sau. “Bridge-Depth Characterizes which Minor-Closed Structural Parameterizations of Vertex Cover Admit a Polynomial Kernel”. In: *SIAM J. Discret. Math.* 36.4 (2022), pp. 2737–2773.

[29] Cornelius Brand, Esra Ceylan, Robert Ganian, Christian Hatschka, and Viktoriia Korchemna. “Edge-Cut Width: An Algorithmically Driven Analogue of Treewidth Based on Edge Cuts”. In: *Graph-Theoretic Concepts in Computer Science - 48th International Workshop, WG 2022, Tübingen, Germany, June 22-24, 2022, Revised Selected Papers*. Ed. by Michael A. Bekos and Michael Kaufmann. Vol. 13453. Lecture Notes in Computer Science. Springer, 2022, pp. 98–113.

[30] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. “Boolean-width of graphs”. In: *Theor. Comput. Sci.* 412.39 (2011), pp. 5187–5204.

[31] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. “Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems”. In: *Theor. Comput. Sci.* 511 (2013), pp. 66–76.

[32] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. “H-join decomposable graphs and algorithms with runtime single exponential in rankwidth”. In: *Discret. Appl. Math.* 158.7 (2010), pp. 809–819.

[33] Jannis Bulian and Anuj Dawar. “Fixed-Parameter Tractable Distances to Sparse Graph Classes”. In: *Algorithmica* 79.1 (2017), pp. 139–158.

[34] Jannis Bulian and Anuj Dawar. “Graph Isomorphism Parameterized by Elimination Distance to Bounded Degree”. In: *Algorithmica* 75.2 (2016), pp. 363–382.

[35] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. “The Complexity of Satisfiability of Small Depth Circuits”. In: *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*. Ed. by Jianer Chen and Fedor V. Fomin. Vol. 5917. Lecture Notes in Computer Science. Springer, 2009, pp. 75–85.

[36] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. “Strong computational lower bounds via parameterized complexity”. In: *J. Comput. Syst. Sci.* 72.8 (2006), pp. 1346–1367.

[37] Li-Hsuan Chen, Felix Reidl, Peter Rossmanith, and Fernando Sánchez Villaamil. “Width, Depth, and Space: Tradeoffs between Branching and Dynamic Programming”. In: *Algorithms* 11.7 (2018), p. 98.

[38] Yijia Chen and Jörg Flum. “FO-Definability of Shrub-Depth”. In: *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*. Ed. by Maribel Fernández and Anca Muscholl. Vol. 152. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 15:1–15:16.

[39] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[40] Derek G. Corneil and Udi Rotics. “On the Relationship Between Clique-Width and Treewidth”. In: *SIAM J. Comput.* 34.4 (2005), pp. 825–847.

[41] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*. Vol. 138. Encyclopedia of mathematics and its applications. Cambridge University Press, 2012.

[42] Bruno Courcelle, Pinar Heggernes, Daniel Meister, Charis Papadopoulos, and Udi Rotics. “A characterisation of clique-width through nested partitions”. In: *Discret. Appl. Math.* 187 (2015), pp. 70–81.

[43] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. “Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width”. In: *Theory Comput. Syst.* 33.2 (2000), pp. 125–150.

[44] Bruno Courcelle and Stephan Olariu. “Upper bounds to the clique width of graphs”. In: *Discret. Appl. Math.* 101.1-3 (2000), pp. 77–114.

[45] Radu Curticapean, Nathan Lindzey, and Jesper Nederlof. “A Tight Lower Bound for Counting Hamiltonian Cycles via Matrix Rank”. In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*. Ed. by Artur Czumaj. SIAM, 2018, pp. 1080–1099.

[46] Marek Cygan, Holger Dell, Daniel Lokshtanov, et al. “On Problems as Hard as CNF-SAT”. In: *ACM Trans. Algorithms* 12.3 (2016), 41:1–41:24.

[47] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, et al. *Parameterized Algorithms*. Springer, 2015.

[48] Marek Cygan, Stefan Kratsch, and Jesper Nederlof. “Fast Hamiltonicity Checking Via Bases of Perfect Matchings”. In: *J. ACM* 65.3 (2018), 12:1–12:46.

[49] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, et al. “Solving Connectivity Problems Parameterized by Treewidth in Single Exponential Time”. In: *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*. Ed. by Rafail Ostrovsky. IEEE Computer Society, 2011, pp. 150–159.

[50] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, et al. “Solving connectivity problems parameterized by treewidth in single exponential time”. In: *CoRR* abs/1103.0534 (2011). arXiv: 1103.0534.

[51] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, et al. “Solving Connectivity Problems Parameterized by Treewidth in Single Exponential Time”. In: *ACM Trans. Algorithms* 18.2 (2022), 17:1–17:31.

[52] Wojciech Czerwinski, Wojciech Nadara, and Marcin Pilipczuk. “Improved Bounds for the Excluded-Minor Approximation of Treedepth”. In: *SIAM J. Discret. Math.* 35.2 (2021), pp. 934–947.

[53] Matt DeVos, O-joung Kwon, and Sang-il Oum. “Branch-depth: Generalizing tree-depth of graphs”. In: *Eur. J. Comb.* 90 (2020), p. 103186.

[54] Josep Díaz, Jordi Petit, and Maria J. Serna. “A survey of graph layout problems”. In: *ACM Comput. Surv.* 34.3 (2002), pp. 313–356.

[55] Reinhard Diestel. *Graph Theory, 4th Edition*. Vol. 173. Graduate texts in mathematics. Springer, 2012.

[56] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[57] Jan Dreier. "Lacon- and Shrub-Decompositions: A New Characterization of First-Order Transductions of Bounded Expansion Classes". In: *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, 2021, pp. 1–13.

[58] Andrew Drucker, Jesper Nederlof, and Rahul Santhanam. "Exponential Time Paradigms Through the Polynomial Time Lens". In: *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*. Ed. by Piotr Sankowski and Christos D. Zaroliagis. Vol. 57. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 36:1–36:14.

[59] Louis Dublois, Michael Lampis, and Vangelis Th. Paschos. "New Algorithms for Mixed Dominating Set". In: *Discret. Math. Theor. Comput. Sci.* 23.1 (2021).

[60] Louis Dublois, Michael Lampis, and Vangelis Th. Paschos. "Upper Dominating Set: Tight algorithms for pathwidth and sub-exponential approximation". In: *Theor. Comput. Sci.* 923 (2022), pp. 271–291.

[61] Guillaume Ducoffe. "Maximum Matching in Almost Linear Time on Graphs of Bounded Clique-Width". In: *Algorithmica* 84.11 (2022), pp. 3489–3520.

[62] Guillaume Ducoffe. "Optimal Centrality Computations Within Bounded Clique-Width Graphs". In: *Algorithmica* 84.11 (2022), pp. 3192–3222.

[63] László Egri, Dániel Marx, and Pawel Rzazewski. "Finding List Homomorphisms from Bounded-treewidth Graphs to Reflexive Graphs: a Complete Complexity Characterization". In: *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*. Ed. by Rolf Niedermeier and Brigitte Vallée. Vol. 96. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 27:1–27:15.

[64] Eduard Eiben, Robert Ganian, Thekla Hamm, Lars Jaffke, and O-joung Kwon. "A Unifying Framework for Characterizing and Computing Width Measures". In: *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*. Ed. by Mark Braverman. Vol. 215. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 63:1–63:23.

[65] Eduard Eiben, Robert Ganian, Thekla Hamm, and O-joung Kwon. "Measuring what matters: A hybrid approach to dynamic programming with treewidth". In: *J. Comput. Syst. Sci.* 121 (2021), pp. 57–75.

[66] Eduard Eiben, Robert Ganian, and Stefan Szeider. "Meta-kernelization using well-structured modulators". In: *Discret. Appl. Math.* 248 (2018), pp. 153–167.

[67] Eduard Eiben, Robert Ganian, and Stefan Szeider. "Solving Problems on Graphs of High Rank-Width". In: *Algorithmica* 80.2 (2018), pp. 742–771.

[68] Baris Can Esmer, Jacob Focke, Dániel Marx, and Pawel Rzazewski. "List homomorphisms by deleting edges and vertices: tight complexity bounds for bounded-treewidth graphs". In: *CoRR* abs/2210.10677 (2022). arXiv: 2210.10677.

[69] Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. "Clique-Width is NP-Complete". In: *SIAM J. Discret. Math.* 23.2 (2009), pp. 909–939.

[70] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[71] Jacob Focke, Dániel Marx, Fionn Mc Inerney, et al. “Tight Complexity Bounds for Counting Generalized Dominating Sets in Bounded-Treewidth Graphs”. In: *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2023, pp. 3664–3683.

[72] Jacob Focke, Dániel Marx, and Pawel Rzazewski. “Counting list homomorphisms from graphs of bounded treewidth: tight complexity bounds”. In: *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*. Ed. by Joseph (Seffi) Naor and Niv Buchbinder. SIAM, 2022, pp. 431–458.

[73] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. “Intractability of Clique-Width Parameterizations”. In: *SIAM J. Comput.* 39.5 (2010), pp. 1941–1956.

[74] Fedor V. Fomin and Tuukka Korhonen. “Fast FPT-approximation of branchwidth”. In: *STOC ’22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*. Ed. by Stefano Leonardi and Anupam Gupta. ACM, 2022, pp. 886–899.

[75] Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. “Planar F-Deletion: Approximation, Kernelization and Optimal FPT Algorithms”. In: *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*. IEEE Computer Society, 2012, pp. 470–479.

[76] Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. “Efficient Computation of Representative Families with Applications in Parameterized and Exact Algorithms”. In: *J. ACM* 63.4 (2016), 29:1–29:60.

[77] Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. “Representative Families of Product Families”. In: *ACM Trans. Algorithms* 13.3 (2017), 36:1–36:29.

[78] Martin Fürer. “Faster Computation of Path-Width”. In: *Combinatorial Algorithms - 27th International Workshop, IWOCA 2016, Helsinki, Finland, August 17-19, 2016, Proceedings*. Ed. by Veli Mäkinen, Simon J. Puglisi, and Leena Salmela. Vol. 9843. Lecture Notes in Computer Science. Springer, 2016, pp. 385–396.

[79] Martin Fürer. “Multi-Clique-Width”. In: *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*. Ed. by Christos H. Papadimitriou. Vol. 67. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 14:1–14:13.

[80] Martin Fürer and Huiwen Yu. “Space Saving by Dynamic Algebraization Based on Tree-Depth”. In: *Theory Comput. Syst.* 61.2 (2017), pp. 283–304.

[81] Jakub Gajarský and Stephan Kreutzer. “Computing Shrub-Depth Decompositions”. In: *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*. Ed. by Christophe Paul and Markus Bläser. Vol. 154. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 56:1–56:17.

[82] Jakub Gajarský, Stephan Kreutzer, Jaroslav Nesetril, et al. “First-Order Interpretations of Bounded Expansion Classes”. In: *ACM Trans. Comput. Log.* 21.4 (2020), 29:1–29:41.

[83] Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. “Parameterized Algorithms for Modular-Width”. In: *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*. Ed. by Gregory Z. Gutin and Stefan Szeider. Vol. 8246. Lecture Notes in Computer Science. Springer, 2013, pp. 163–176.

[84] Tibor Gallai. “Transitiv orientierbare graphen”. In: *Acta Mathematica Hungarica* 18.1-2 (1967), pp. 25–66.

[85] Robert Ganian. “Using Neighborhood Diversity to Solve Hard Problems”. In: *CoRR* abs/1201.3091 (2012). arXiv: 1201.3091.

[86] Robert Ganian, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. “The Fine-Grained Complexity of Graph Homomorphism Parameterized by Clique-Width”. In: *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*. Ed. by Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 66:1–66:20.

[87] Robert Ganian, Petr Hlinený, Jaroslav Nesetril, Jan Obdrzálek, and Patrice Ossona de Mendez. “Shrub-depth: Capturing Height of Dense Graphs”. In: *Log. Methods Comput. Sci.* 15.1 (2019).

[88] Robert Ganian, Petr Hlinený, Jaroslav Nesetril, et al. “When Trees Grow Low: Shrubs and Fast MSO1”. In: *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*. Ed. by Branislav Rovan, Vladimiro Sassone, and Peter Widmayer. Vol. 7464. Lecture Notes in Computer Science. Springer, 2012, pp. 419–430.

[89] Robert Ganian and Viktoriia Korchemna. “Slim Tree-Cut Width”. In: *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*. Ed. by Holger Dell and Jesper Nederlof. Vol. 249. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 15:1–15:18.

[90] Robert Ganian, Friedrich Slivovsky, and Stefan Szeider. “Meta-kernelization with structural parameters”. In: *J. Comput. Syst. Sci.* 82.2 (2016), pp. 333–346.

[91] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[92] M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. “Some Simplified NP-Complete Graph Problems”. In: *Theor. Comput. Sci.* 1.3 (1976), pp. 237–267.

[93] William I. Gasarch. “Guest Column: The Third P=?NP Poll”. In: *SIGACT News* 50.1 (2019), 38–59.

[94] Bas A. M. van Geffen, Bart M. P. Jansen, Arnoud A. W. M. de Kroon, and Rolf Morel. “Lower Bounds for Dynamic Programming on Planar Graphs of Bounded Cutwidth”. In: *J. Graph Algorithms Appl.* 24.3 (2020), pp. 461–482.

[95] Carla Groenland, Isja Mannens, Jesper Nederlof, and Krisztina Szilágyi. “Tight Bounds for Counting Colorings and Connected Edge Sets Parameterized by Cutwidth”. In: *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*. Ed. by Petra Berenbrink and Benjamin Monmege. Vol. 219. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 36:1–36:20.

[96] Jiong Guo and Danny Hermelin, eds. *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*. Vol. 63. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[97] Michel Habib and Christophe Paul. “A survey of the algorithmic aspects of modular decomposition”. In: *Comput. Sci. Rev.* 4.1 (2010), pp. 41–59.

[98] György Hajós. “Über eine Konstruktion nicht nn-färbbarer Graphen”. In: *Wiss. Z. Martin Luther Univ. Halle-Wittenberg Math.-Natur. Reihe* 10 (1961), pp. 116–117.

[99] Falko Hegerfeld and Stefan Kratsch. “Solving Connectivity Problems Parameterized by Treedepth in Single-Exponential Time and Polynomial Space”. In: *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*. Ed. by Christophe Paul and Markus Bläser. Vol. 154. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 29:1–29:16.

[100] Falko Hegerfeld and Stefan Kratsch. “Tight algorithms for connectivity problems parameterized by clique-width”. In: *CoRR* abs/2302.03627 (2023). arXiv: 2302.03627.

[101] Falko Hegerfeld and Stefan Kratsch. “Tight Algorithms for Connectivity Problems Parameterized by Modular-Treewidth”. In: *CoRR* abs/2302.14128 (2023). arXiv: 2302.14128.

[102] Falko Hegerfeld and Stefan Kratsch. “Towards Exact Structural Thresholds for Parameterized Complexity”. In: *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*. Ed. by Holger Dell and Jesper Nederlof. Vol. 249. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 17:1–17:20.

[103] Eva-Maria C. Hols, Stefan Kratsch, and Astrid Pieterse. “Elimination Distances, Blocking Sets, and Kernels for Vertex Cover”. In: *SIAM J. Discret. Math.* 36.3 (2022), pp. 1955–1990.

[104] Russell Impagliazzo and Ramamohan Paturi. “On the Complexity of k-SAT”. In: *J. Comput. Syst. Sci.* 62.2 (2001), pp. 367–375.

[105] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. “Which Problems Have Strongly Exponential Complexity?” In: *J. Comput. Syst. Sci.* 63.4 (2001), pp. 512–530.

[106] Yoichi Iwata and Yuichi Yoshida. “On the Equivalence among Problems of Bounded Width”. In: *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*. Ed. by Nikhil Bansal and Irene Finocchi. Vol. 9294. Lecture Notes in Computer Science. Springer, 2015, pp. 754–765.

[107] Ashwin Jacob, Fahad Panolan, Venkatesh Raman, and Vibha Sahlot. “Structural Parameterizations with Modulator Oblivion”. In: *Algorithmica* 84.8 (2022), pp. 2335–2357.

[108] Hugo Jacob, Thomas Bellitto, Oscar Defrain, and Marcin Pilipczuk. “Close Relatives (Of Feedback Vertex Set), Revisited”. In: *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*. Ed. by Petr A. Golovach and Meirav Zehavi. Vol. 214. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 21:1–21:15.

[109] Lars Jaffke and Bart M. P. Jansen. “Fine-grained parameterized complexity analysis of graph coloring problems”. In: *Discret. Appl. Math.* 327 (2023), pp. 33–46.

[110] Bart M. P. Jansen, Jari J. H. de Kroon, and Michal Wlodarczyk. “Vertex deletion parameterized by elimination distance and even less”. In: *STOC ’21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*. Ed. by Samir Khuller and Virginia Vassilevska Williams. ACM, 2021, pp. 1757–1769.

[111] Bart M. P. Jansen and Jesper Nederlof. “Computing the chromatic number using graph decompositions via matrix rank”. In: *Theor. Comput. Sci.* 795 (2019), pp. 520–539.

[112] Ojvind Johansson. “Clique-decomposition, NLC-decomposition, and modular decomposition-relationships and results for random graphs”. In: *Congressus Numerantium* (1998), pp. 39–60.

[113] Meir Katchalski, William McCuaig, and Suzanne M. Seager. “Ordered colourings”. In: *Discret. Math.* 142.1-3 (1995), pp. 141–154.

[114] Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. “Structural parameters, tight bounds, and approximation for (k, r)-center”. In: *Discrete Applied Mathematics* 264 (2019), pp. 90–117.

[115] Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. “Structurally parameterized d-scattered set”. In: *Discret. Appl. Math.* 308 (2022), pp. 168–186.

[116] Ton Kloks. *Treewidth, Computations and Approximations*. Vol. 842. Lecture Notes in Computer Science. Springer, 1994.

[117] Tuukka Korhonen. “A Single-Exponential Time 2-Approximation Algorithm for Treewidth”. In: *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*. IEEE, 2021, pp. 184–192.

[118] Tuukka Korhonen and Daniel Lokshtanov. “An Improved Parameterized Algorithm for Treewidth”. In: *CoRR* abs/2211.07154 (2022). To appear at STOC 2023. arXiv: 2211.07154.

[119] Stefan Kratsch and Florian Nelles. “Efficient and Adaptive Parameterized Algorithms on Modular Decompositions”. In: *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*. Ed. by Yossi Azar, Hannah Bast, and Grzegorz Herman. Vol. 112. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 55:1–55:15.

[120] Stefan Kratsch and Florian Nelles. “Efficient parameterized algorithms on graphs with heterogeneous structure: Combining tree-depth and modular-width”. In: *CoRR* abs/2209.14429 (2022). arXiv: 2209.14429.

[121] O-joung Kwon, Rose McCarty, Sang-il Oum, and Paul Wollan. “Obstructions for bounded shrub-depth and rank-depth”. In: *J. Comb. Theory, Ser. B* 149 (2021), pp. 76–91.

[122] Michael Lampis. “Algorithmic Meta-theorems for Restrictions of Treewidth”. In: *Algorithmica* 64.1 (2012), pp. 19–37.

[123] Michael Lampis. “Finer Tight Bounds for Coloring on Clique-Width”. In: *SIAM J. Discret. Math.* 34.3 (2020), pp. 1538–1558.

[124] John M. Lewis and Mihalis Yannakakis. “The Node-Deletion Problem for Hereditary Properties is NP-Complete”. In: *J. Comput. Syst. Sci.* 20.2 (1980), pp. 219–230.

[125] Jason Li and Jesper Nederlof. “Detecting Feedback Vertex Sets of Size k in $O^\star$ (2.7k) Time”. In: *ACM Trans. Algorithms* 18.4 (2022), 34:1–34:26.

[126] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. “Known Algorithms on Graphs of Bounded Treewidth Are Probably Optimal”. In: *ACM Trans. Algorithms* 14.2 (2018), 13:1–13:30.

[127] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. “Lower bounds based on the Exponential Time Hypothesis”. In: *Bull. EATCS* 105 (2011), pp. 41–72.

[128] Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. “Faster Parameterized Algorithms Using Linear Programming”. In: *ACM Trans. Algorithms* 11.2 (2014), 15:1–15:31.

[129] Daniel Lokshtanov and Jesper Nederlof. “Saving space by algebraization”. In: *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*. Ed. by Leonard J. Schulman. ACM, 2010, pp. 321–330.

[130] Loïc Magne, Christophe Paul, Abhijat Sharma, and Dimitrios M. Thilikos. “Edge-trewidth: Algorithmic and combinatorial properties”. In: *CoRR* abs/2112.07524 (2021). arXiv: 2112.07524.

[131] Dániel Marx, Govind S. Sankar, and Philipp Schepper. “Anti-Factor Is FPT Parameterized by Treewidth and List Size (But Counting Is Hard)”. In: *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*. Ed. by Holger Dell and Jesper Nederlof. Vol. 249. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 22:1–22:23.

[132] Dániel Marx, Govind S. Sankar, and Philipp Schepper. “Degrees and Gaps: Tight Complexity Results of General Factor Problems Parameterized by Treewidth and Cutwidth”. In: *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*. Ed. by Nikhil Bansal, Emanuela Merelli, and James Worrell. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 95:1–95:20.

[133] Stefan Mengel. “Parameterized Compilation Lower Bounds for Restricted CNF-Formulas”. In: *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*. Ed. by Nadia Creignou and Daniel Le Berre. Vol. 9710. Lecture Notes in Computer Science. Springer, 2016, pp. 3–12.

[134] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. “Matching is as easy as matrix inversion”. In: *Combinatorica* 7.1 (1987), pp. 105–113.

[135] Wojciech Nadara, Michal Pilipczuk, and Marcin Smulewicz. “Computing Treedepth in Polynomial Space and Linear FPT Time”. In: *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*. Ed. by Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman. Vol. 244. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 79:1–79:14.

[136] Jesper Nederlof. “Algorithms for NP-Hard Problems via Rank-Related Parameters of Matrices”. In: *Treewidth, Kernels, and Algorithms - Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*. Ed. by Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen. Vol. 12160. Lecture Notes in Computer Science. Springer, 2020, pp. 145–164.

[137] Jesper Nederlof, Michal Pilipczuk, Céline M. F. Swennenhuis, and Karol Wegrzycki. “Hamiltonian Cycle Parameterized by Treedepth in Single Exponential Time and Polynomial Space”. In: *Graph-Theoretic Concepts in Computer Science - 46th International Workshop, WG 2020, Leeds, UK, June 24-26, 2020, Revised Selected Papers*. Ed. by Isolde Adler and Haiko Müller. Vol. 12301. Lecture Notes in Computer Science. Springer, 2020, pp. 27–39.

[138] Jesper Nederlof, Michal Pilipczuk, Céline M. F. Swennenhuis, and Karol Wegrzycki. “Isolation Schemes for Problems on Decomposable Graphs”. In: *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*. Ed. by Petra Berenbrink and Benjamin Monmege. Vol. 219. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 50:1–50:20.

[139] Jesper Nederlof and Johan M. M. van Rooij. “Inclusion/Exclusion Branching for Partial Dominating Set and Set Splitting”. In: *Parameterized and Exact Computation - 5th International Symposium, IPEC 2010, Chennai, India, December 13-15, 2010. Proceedings*. Ed. by Venkatesh Raman and Saket Saurabh. Vol. 6478. Lecture Notes in Computer Science. Springer, 2010, pp. 204–215.

[140] Jesper Nederlof, Johan M. M. van Rooij, and Thomas C. van Dijk. “Inclusion/Exclusion Meets Measure and Conquer”. In: *Algorithmica* 69.3 (2014), pp. 685–740.

[141] Jaroslav Nesetril and Patrice Ossona de Mendez. “On Low Tree-Depth Decompositions”. In: *Graphs Comb.* 31.6 (2015), pp. 1941–1963.

[142] Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*. Vol. 28. Algorithms and combinatorics. Springer, 2012.

[143] Jaroslav Nesetril and Patrice Ossona de Mendez. "Tree-depth, subgraph coloring and homomorphism bounds". In: *Eur. J. Comb.* 27.6 (2006), pp. 1022–1041.

[144] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[145] Karolina Okrasa, Marta Piecyk, and Pawel Rzazewski. "Full Complexity Classification of the List Homomorphism Problem for Bounded-Treewidth Graphs". In: *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*. Ed. by Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders. Vol. 173. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 74:1–74:24.

[146] Karolina Okrasa and Pawel Rzazewski. "Fine-Grained Complexity of the Graph Homomorphism Problem for Bounded-Treewidth Graphs". In: *SIAM J. Comput.* 50.2 (2021), pp. 487–508.

[147] Sang-il Oum. "Approximating rank-width and clique-width quickly". In: *ACM Trans. Algorithms* 5.1 (2008), 10:1–10:20.

[148] Sang-il Oum. "Rank-width is less than or equal to branch-width". In: *J. Graph Theory* 57.3 (2008), pp. 239–244.

[149] Sang-il Oum and Paul D. Seymour. "Approximating clique-width and branch-width". In: *J. Comb. Theory, Ser. B* 96.4 (2006), pp. 514–528.

[150] Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. "Model Counting for CNF Formulas of Bounded Modular Treewidth". In: *Algorithmica* 76.1 (2016), pp. 168–194.

[151] Marta Piecyk and Pawel Rzazewski. "Fine-Grained Complexity of the List Homomorphism Problem: Feedback Vertex Set and Cutwidth". In: *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*. Ed. by Markus Bläser and Benjamin Monmege. Vol. 187. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 56:1–56:17.

[152] Michal Pilipczuk. "Problems Parameterized by Treewidth Tractable in Single Exponential Time: A Logical Approach". In: *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*. Ed. by Filip Murlak and Piotr Sankowski. Vol. 6907. Lecture Notes in Computer Science. Springer, 2011, pp. 520–531.

[153] Michal Pilipczuk, Patrice Ossona de Mendez, and Sebastian Siebertz. "Transducing paths in graph classes with unbounded shrubdepth". In: *CoRR* abs/2203.16900 (2022). arXiv: 2203.16900.

[154] Michal Pilipczuk and Sebastian Siebertz. "Polynomial bounds for centered colorings on proper minor-closed graph classes". In: *J. Comb. Theory, Ser. B* 151 (2021), pp. 111–147.

[155] Michal Pilipczuk and Marcin Wrochna. "On Space Efficiency of Algorithms Working on Structural Decompositions of Graphs". In: *ACM Trans. Comput. Theory* 9.4 (2018), 18:1–18:36.

[156] Willem J. A. Pino, Hans L. Bodlaender, and Johan M. M. van Rooij. "Cut and Count and Representative Sets on Branch Decompositions". In: *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*. Ed. by Jiong Guo and Danny Hermelin. Vol. 63. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 27:1–27:12.

[157] Alex Pothen. “The complexity of optimal elimination trees”. In: *Technical Report* (1988).

[158] Michaël Rao. “Clique-width of graphs defined by one-vertex extensions”. In: *Discret. Math.* 308.24 (2008), pp. 6157–6165.

[159] Michaël Rao. “Décompositions de graphes et algorithmes efficaces”. Theses. Université Paul Verlaine - Metz, June 2006.

[160] Neil Robertson and Paul D. Seymour. “Graph Minors. II. Algorithmic Aspects of Tree-Width”. In: *J. Algorithms* 7.3 (1986), pp. 309–322.

[161] Neil Robertson and Paul D. Seymour. “Graph minors. X. Obstructions to tree-decomposition”. In: *J. Comb. Theory, Ser. B* 52.2 (1991), pp. 153–190.

[162] Johan M. M. van Rooij. “A Generic Convolution Algorithm for Join Operations on Tree Decompositions”. In: *Computer Science - Theory and Applications - 16th International Computer Science Symposium in Russia, CSR 2021, Sochi, Russia, June 28 - July 2, 2021, Proceedings*. Ed. by Rahul Santhanam and Daniil Musatov. Vol. 12730. Lecture Notes in Computer Science. Springer, 2021, pp. 435–459.

[163] Johan M. M. van Rooij, Hans L. Bodlaender, Erik Jan van Leeuwen, Peter Rossmanith, and Martin Vatshelle. “Fast Dynamic Programming on Graph Decompositions”. In: *CoRR* abs/1806.01667 (2018). arXiv: 1806.01667.

[164] Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. “Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution”. In: *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*. Ed. by Amos Fiat and Peter Sanders. Vol. 5757. Lecture Notes in Computer Science. Springer, 2009, pp. 566–577.

[165] Sigve Hortemo Sæther and Jan Arne Telle. “Between Treewidth and Clique-Width”. In: *Algorithmica* 75.1 (2016), pp. 218–253.

[166] Paul D. Seymour and Robin Thomas. “Graph Searching and a Min-Max Theorem for Tree-Width”. In: *J. Comb. Theory, Ser. B* 58.1 (1993), pp. 22–33.

[167] Noam Ta-Shma. “A simple proof of the Isolation Lemma”. In: *Electron. Colloquium Comput. Complex.* 22 (2015), p. 80.

[168] Atsushi Takahashi, Shuichi Ueno, and Yoji Kajitani. “Mixed Searching and Proper-Path-Width”. In: *Theor. Comput. Sci.* 137.2 (1995), pp. 253–268.

[169] Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. “Simpler Linear-Time Modular Decomposition Via Recursive Factorizing Permutations”. In: *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*. Ed. by Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, et al. Vol. 5125. Lecture Notes in Computer Science. Springer, 2008, pp. 634–645.

[170] Jan Arne Telle and Andrzej Proskurowski. “Practical Algorithms on Partial k-Trees with an Application to Domination-like Problems”. In: *Algorithms and Data Structures, Third Workshop, WADS ’93, Montréal, Canada, August 11-13, 1993, Proceedings*. Ed. by Frank K. H. A. Dehne, Jörg-Rüdiger Sack, Nicola Santoro, and Sue Whitesides. Vol. 709. Lecture Notes in Computer Science. Springer, 1993, pp. 610–621.

[171] Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. “Cutwidth I: A linear time fixed parameter algorithm”. In: *J. Algorithms* 56.1 (2005), pp. 1–24.

[172] Martin Vatshelle. “New width parameters of graphs”. In: *Doctoral Dissertation, The University of Bergen* (2012).

[173] Egon Wanke. “k-NLC Graphs and Polynomial Algorithms”. In: *Discret. Appl. Math.* 54.2-3 (1994), pp. 251–266.

[174] Michal Wlodarczyk. “Clifford Algebras Meet Tree Decompositions”. In: *Algorithmica* 81.2 (2019), pp. 497–518.

[175] Gerhard J. Woeginger. “Space and Time Complexity of Exact Algorithms: Some Open Problems (Invited Talk)”. In: *Parameterized and Exact Computation, First International Workshop, IWPEC 2004, Bergen, Norway, September 14-17, 2004, Proceedings*. Ed. by Rodney G. Downey, Michael R. Fellows, and Frank K. H. A. Dehne. Vol. 3162. Lecture Notes in Computer Science. Springer, 2004, pp. 281–290.

[176] Paul Wollan. “The structure of graphs not admitting a fixed immersion”. In: *J. Comb. Theory, Ser. B* 110 (2015), pp. 47–66.

[177] Michal Ziobro and Marcin Pilipczuk. “Finding Hamiltonian Cycle in Graphs of Bounded Treewidth: Experimental Evaluation”. In: *ACM J. Exp. Algorithmics* 24.1 (2019), 2.7:1–2.7:18.

Appendix

A

A.1 Problem Definitions

In this section, we define all problems appearing in this thesis. Usually, we denote cost/objective functions by $\mathbf{c}$, the cost/cardinality constraint by $\overline{b}$ (short for *budget*), and solution sets are denoted by X.

A.1.1 Graph Problems

Vertex Cover

Input: An undirected graph $G = (V, E)$, a cost function $\mathbf{c}\colon V \to \mathbb{N} \setminus \{0\}$ and an integer $\overline{b}$.

Question: Is there a set $X \subseteq V$, $\mathbf{c}(X) \leq \overline{b}$, such that $G - X$ contains no edges?

Independent Set

Input: An undirected graph $G = (V, E)$, a cost function $\mathbf{c}\colon V \to \mathbb{N} \setminus \{0\}$ and an integer $\overline{b}$.

Question: Is there a set $X \subseteq V$, $\mathbf{c}(X) \geq \overline{b}$, such that $G[X]$ contains no edges?

Odd Cycle Transversal

Input: An undirected graph $G = (V, E)$, a cost function $\mathbf{c}\colon V \to \mathbb{N} \setminus \{0\}$ and an integer $\overline{b}$.

Question: Is there a set $X \subseteq V$, $|X| \leq \overline{b}$, such that $G - X$ is bipartite?

Dominating Set

Input: An undirected graph $G = (V, E)$, a cost function $\mathbf{c}\colon V \to \mathbb{N} \setminus \{0\}$ and an integer $\overline{b}$.

Question: Is there a set $X \subseteq V$, $\mathbf{c}(X) \leq \overline{b}$, such that $N[X] = V$?

Total Dominating Set

Input: An undirected graph $G = (V, E)$, a cost function $\mathbf{c}\colon V \to \mathbb{N} \setminus \{0\}$ and an integer $\overline{b}$.

Question: Is there a set $X \subseteq V$, $\mathbf{c}(X) \leq \overline{b}$, such that $N(v) \cap X \neq \emptyset$ for all $v \in V$?

q-Coloring

Input: An undirected graph $G = (V, E)$.

Question: Is $\chi(G) \leq q$?

List q-Coloring

Input: An undirected graph $G = (V, E)$ and lists $\Lambda(v) \subseteq [q]$ for all $v \in V$.
Question: Is there a q-coloring $\varphi\colon V \to [q]$ of G usch that $\varphi(v) \in \Lambda(v)$ for all $v \in V$?

Deletion to q-Colorable

Input: An undirected graph $G = (V, E)$, a cost function $\mathbf{c}\colon V \to \mathbb{N}_{>0}$ and an integer $\overline{b} \in \mathbb{N}$.
Question: Is there a set $X \subseteq V$, $\mathbf{c}(X) \leq \overline{b}$, such that $G - X$ is q-colorable?

Maximum Cut

Input: An undirected graph $G = (V, E)$ and an integer $\overline{b}$.
Question: Is there a set $X \subseteq V$, such that $|\{\{u, v\} \in E : u \in X, v \notin X\}| \geq \overline{b}$?

H-free Deletion

Input: An undirected graph $G = (V, E)$ and an integer $\overline{b}$.
Question: Is there a set $X \subseteq V$, $|X| \leq \overline{b}$, such that $G - X$ contains no subgraph isomorphic to H?

A.1.2 Connectivity Problems

(Node) Steiner Tree

Input: An undirected graph $G = (V, E)$, a set of terminals $K \subseteq V$, a cost function $\mathbf{c}\colon V \to \mathbb{N} \setminus \{0\}$ and an integer $\overline{b}$.
Question: Is there a set $X \subseteq V$, $\mathbf{c}(X) \leq \overline{b}$, such that $K \subseteq X$ and $G[X]$ is connected?

Connected Vertex Cover

Input: An undirected graph $G = (V, E)$, a cost function $\mathbf{c}\colon V \to \mathbb{N} \setminus \{0\}$ and an integer $\overline{b}$.
Question: Is there a set $X \subseteq V$, $\mathbf{c}(X) \leq \overline{b}$, such that $G - X$ contains no edges and $G[X]$ is connected?

Connected Dominating Set

Input: An undirected graph $G = (V, E)$, a cost function $\mathbf{c}\colon V \to \mathbb{N} \setminus \{0\}$ and an integer $\overline{b}$.
Question: Is there a set $X \subseteq V$, $\mathbf{c}(X) \leq \overline{b}$, such that $N[X] = V$ and $G[X]$ is connected?

Connected Odd Cycle Transversal

Input: An undirected graph $G = (V, E)$, a cost function $\mathbf{c}\colon V \to \mathbb{N}_{>0}$ and an integer $\overline{b} \in \mathbb{N}$.
Question: Is there a set $X \subseteq V$, $\mathbf{c}(X) \leq \overline{b}$, such that $G[X]$ is connected and $G - X$ is 2-colorable?

Connected Deletion to q-Colorable

Input: An undirected graph $G = (V, E)$, a cost function $\mathbf{c}\colon V \to \mathbb{N}_{>0}$ and an integer $\overline{b} \in \mathbb{N}$.

Question: Is there a set $X \subseteq V$, $\mathbf{c}(X) \leq \overline{b}$, such that $G[X]$ is connected and $G - X$ is q-colorable?

Feedback Vertex Set

Input: An undirected graph $G = (V, E)$, a cost function $\mathbf{c}\colon V \to \mathbb{N} \setminus \{0\}$ and an integer $\overline{b}$.

Question: Is there a set $X \subseteq V$, $\mathbf{c}(X) \leq \overline{b}$, such that $G - X$ contains no cycles?

Induced Forest

Input: An undirected graph $G = (V, E)$, a cost function $\mathbf{c}\colon V \to \mathbb{N} \setminus \{0\}$ and an integer $\overline{b}$.

Question: Is there a set $X \subseteq V$, $\mathbf{c}(X) \geq \overline{b}$, such that $G[X]$ contains no cycles?

A.1.3 Satisfiability and Hitting Set

Satisfiability

Input: A boolean formula σ in conjunctive normal form.

Question: Is there a satisfying assignment τ for σ?

q-Satisfiability

Input: A boolean formula σ in conjunctive normal form with clauses of size at most q.

Question: Is there a satisfying assignment τ for σ?

q-Hitting Set

Input: A universe U and a set family $\mathcal{F}$ over U of sets of size at most q and an integer $\overline{h}$.

Question: Is there a set $H \subseteq U$, $|H| \leq \overline{h}$, such that $H \cap S \neq \emptyset$ for all $S \in \mathcal{F}$?

Selbstständigkeitserklärung

„Ich erkläre, dass ich die Dissertation selbständig und nur unter Verwendung der von mir gemäß § 7 Abs. 3 der Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät, veröffentlicht im Amtlichen Mitteilungsblatt der Humboldt-Universität zu Berlin Nr. 42/2018 am 11.07.2018 angegebenen Hilfsmittel angefertigt habe."

Berlin, 24.04.2023

Falko Hegerfeld