Parameterized Graph Modification Algorithms

Pål Grønås Drange

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

2015

Scientific environment

The work of this thesis was done in the algorithms group at the *Department of Informatics* at the University of Bergen. I was hosted by the ICT Research School.

This project has been done completely under the supervision of Professor Fedor V. Fomin, and the project has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013), ERC Grant Agreement n. 267959, *Rigorous Theory of Preprocessing*.

Acknowledgments

First and foremost I would like to thank my supervisor Professor Fedor V. Fomin. Needless to say, this project would never have been conducted had it not been for the patience, insight and foresight that you had in the beginning of this project. Most of all, I am deeply grateful for all the conversations we had, and for your time. I never met a closed door, and regardless of how busy you were, you always seemed to have time for me when I needed it.

Second, I would like to thank my family for their patience, especially this last year. This thesis is of course not a result of my own, nor even the collaboration with my supervisor. Without you, I would be nothing. Thank you, Hannah, for your support. I would like to thank Erik Parmann for dragging me through my first five and a half year at the university. Without you around, the days, and indeed my life as a student would be grey and dull. To all my friends, all three of them, Mats, Kenneth, and Espen. Thanks for being cool.

There's a saying that goes like "if you're always the smartest person in the room, you're in the wrong room." However, I think equivalently there should be a saying (there is now) that goes like "if you're always the slowest person in the room, you're in the wrong room." That being said, I have always felt welcome, and having been given the opportunity to work with the people in the algorithm group at the University of Bergen, has been a very humbling experience. You are the greatest guys in town, and I will never forget these years. Especially, I would like to thank Markus for all the traveling we did together, not to mention the wedding and all our projects, Pim for all the chats we had, and for the help you gave me whenever I needed, Michał for having been my academic big brother and teaching me everything, and Jan Arne for the organization of the Turing centennial and Eurocomb. I would also like to thank all of my coauthors. A big thanks goes to the logic group of Bergen led by Thomas Ågotnes. Thank you for guiding me towards research! Thanks to Truls, Sjur, Paul Simon, and Piotr. Felix, Hannah, and (obviously) Fedor read and provided valuable feedback on preliminary drafts of this thesis, and if this thesis contains anything nice at all, it is in big part thanks to you. I owe you all.

I would finally like to thank my thesis committee Hadas Shachnai, Magnus Wahlström, and Jan Arne Telle for taking the time to read and evaluate the thesis.

Abstract

Graph modification problems form an important class of algorithmic problems in computer science. In this thesis, we study edge modification problems towards classes related to chordal graphs, with the main focus on trivially perfect graphs and threshold graphs. We provide several new results in classical complexity, kernelization complexity, and subexponential parameterized complexity. In all cases we give positive and negative results—giving polynomial time algorithms as well as NP-hardness results, polynomial kernels as well as polynomial kernel impossibility results, and we give subexponential time algorithms, and show that many problems do not admit such algorithms unless the exponential time hypothesis fails.

Our main focus is on the subexponential time complexity of edge modification problems. For that to make sense, we first need to figure out whether or not we actually need super-polynomial time. We show that editing towards trivially perfect graphs, threshold graphs, and chain graphs are all NP-complete, resolving 15 year old open questions. When a problem is shown to be NP-complete, we study exactly how much exponential time is needed for an algorithm to solve it. We provide several subexponential time algorithms, for, e.g., editing towards chain graphs and threshold graphs, as well as completing towards trivially perfect graphs. We complement our results by showing that small alterations in the target graph classes yields much harder problems: Editing towards trivially perfect graphs and cographs is not possible in subexponential time unless the exponential time hypothesis fails.

A first step in our subexponential time algorithms, and an otherwise natural first step in dealing with NP-hard problems is offered by the toolbox of polynomial kernelization. In polynomial kernelizations, we are asked to design polynomial time compression algorithms that shrink the input instances to output instances bounded polynomially in a yes-solution. We provide polynomial kernels for all edge modification problems towards trivially perfect graphs, threshold graphs and chain graphs. In addition, we show that on bounded degree input graphs, we obtain polynomial kernels for any editing or deletion problem towards graph classes characterizable by a finite set of forbidden induced subgraphs. Finally, we show that we should not expect the same result for completion problems by proving that such a compression algorithm would imply the collapse of the polynomial hierarchy.

List of papers

The results of this thesis are in part based on the following publications:

1. Exploring the Subexponential Parameterized Complexity of Completion Problems [DFPV15].
Pål Grønås Drange, Fedor V. Fomin, Michał Pilipczuk, and Yngve Villanger. *In ACM Transactions on Computation Theory 7(4):Article 14, 2015. A preliminary version of this article was presented at* STACS 2014 *[DFPV14].*

> Chapters 10 and 11 of Part III, the subexponential time complexity of completing *trivially perfect graphs* and *pseudosplit graphs*, respectively, as well as the lower bounds result of Chapters 15 Part IV is based on results from this work. Some properties of trivially perfect graphs discussed in Section 2.2 are taken from this publication.

2. A Polynomial Kernel for Trivially Perfect Editing [DP15].
Pål Grønås Drange and Michał Pilipczuk. *In* ESA 2015*, volume 9294 of* Lecture Notes in Computer Science*, pages 411–423. Springer, 2015.*

> Chapter 6 in Part II and Chapter 14 in Part IV are based on results from this work.

3. On the Threshold of Intractability [DDLS15].
Pål Grønås Drange, Markus Dregi, Daniel Lokshtanov, and Blair D. Sullivan. *In* ESA 2015*, volume 9294 of* Lecture Notes in Computer Science*, pages 424–436. Springer, 2015.*

> Chapters 4 and 5 in Part II, Chapter 9 in Part III, and Chapter 13 in Part IV are based on results from this work.

4. Fast Biclustering by Dual Parameterization [DRSVS15].
Pål Grønås Drange, Felix Reidl, Fernando Sánchez Villaamil, and Somnath Sikdar. *In* IPEC 2015*, to appear, 2015.*

> Chapter 7 in Part II and Chapters 12 and 16 from Parts III and IV are based on results from this work.

5. Compressing bounded degree graphs [DDS15].
Pål Grønås Drange, Markus Dregi, and R.D. Sandeep. *Submitted.*

> Chapter 8 from Part II is based on results from this work.

Contents

List of Figures

Part I

Introduction and preliminaries

Chapter 1

Introduction and motivation

1.1 Graph modification

A mixture of colleagues, family, old friends and new, will be attending dinner at your house tonight. Your boss is also coming. Anticipating poor social chemistry, you arrive at the conclusion that the perfect seating arrangement is paramount. Individuals who like each other should be seated together and individuals who do not know each other, should not. Not knowing who are friends with whom, you log in to Generic, the most popular social network platform. Meticulously, you scribble down all the friendships among your guests, and end up with a nice little drawing.

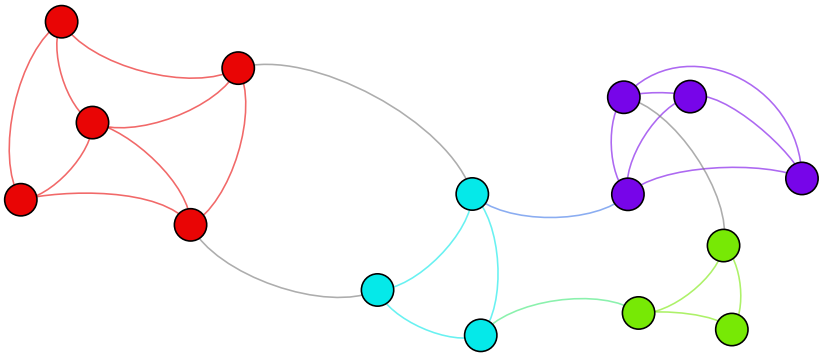


Figure 1.1: The dinner party

With each guest drawn as a round little dot, you draw a line between two guests if they are friends on Generic. Not writing the names of the individuals representing the dots helps prevent bias. You find what seems to be four people that are all mutually friends but do not have that many other friends. You color them purple and put them aside. Thereafter, you find a small triangle in the remainder with similar properties. You color them green. After a while you have colored all your guests and found your final seating arrangement. Being mathematically inclined, you begin to wonder over this phenomenon. What were you actually looking for

just now? You realize that you wanted to minimize the number of friends that you separate, whilst simultaneously minimizing the number of non-friends you place at the same table.

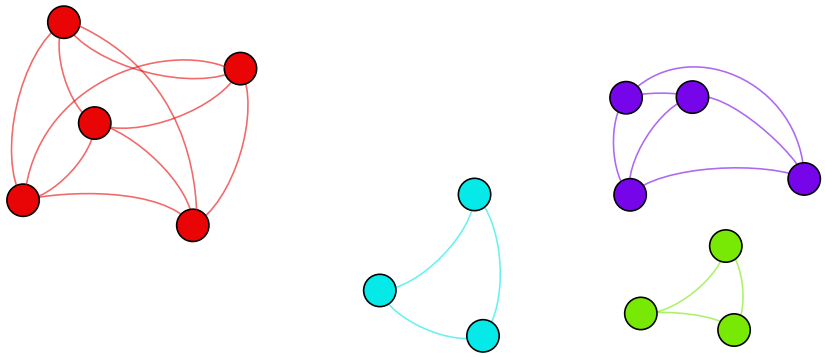


Figure 1.2: The seating arrangement

Throughout this thesis, for historical reasons, we will refer to the guests as V (the dots in your drawing). This letter stands for *vertex* (plural *vertices*). We refer to the friendships with the letter E, for *edge*. The two pictures are visualizations of two different graphs, and we denote the graphs with the letter G.

This seating arrangement problem was a quite simple problem, but still, it makes you think; was the seating arrangement optimal? Or was there one separating fewer friends at the same time joining fewer non-friends? This problem is well-studied in computer science, not necessarily purely because of the applications to dinner parties. The problem in the general sense goes by the name Cluster Editing[1], and has far-reaching applications in biology, computer science, economy, and social sciences to name a few.

The reason for the name Cluster Editing is quite simple. The *cluster* part signifies that the "target graph", the type of end result we are looking for, is a perfect clustering. A perfect clustering here means that everyone sitting at the same table are friends, and that no two friends are sitting at different tables. The *editing* part conveys that we are allowed to both add and delete friendships to obtain the seating arrangements. As we will see later, these "operations"—adding and deleting friendships—correspond to false negatives and false positives. We want to minimize the number of both these errors.

This thesis is about these kinds of problems and how to optimally solve them. It might seem trivial at first to solve such an instance optimally, but this was a very small dinner party. In Figure 1.3, we have a slightly bigger network, and as we can see, things get rather messy. And this network is still pretty small, it's just a portion of the collaboration network of *one* person. If we take the collaboration network of all active researchers in computer science, we would have a network 10 000 times bigger. And it would still be called a small network.

[1]The formal problem names will throughout this thesis be typeset in *small caps*.

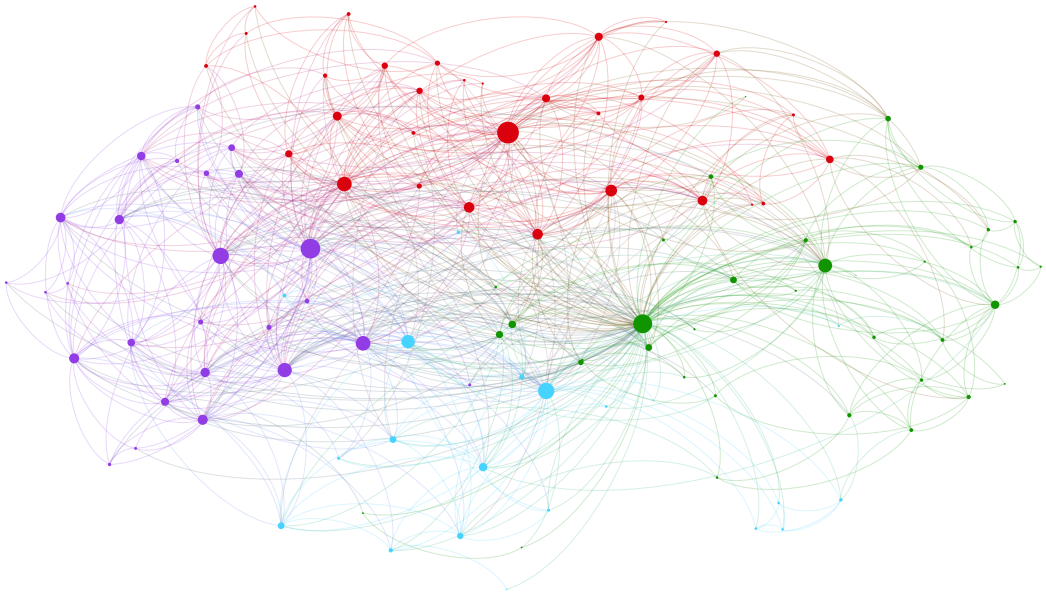


Figure 1.3: A social network

Later in this thesis we will discuss all combinations of modifying a graph. In addition to editing, we study *deletion*, where we only allow to delete friendships, and *completion*, where we only allow to add new friendships.

Social networks and applications. In an abundance of cases in natural and social sciences, as well as in computer and technology science fields, our data is of this type of relational form. If our model is based on elements that are pairwise related, we have a graph at hand. Examples of such models are correlation graphs, every sort of transportation and communication network, electronic circuits, chemical bonds, neural networks, and as we saw above, social networks. A social network is any network that originates from social interaction, with examples ranging from communication between employees in a company, friendship networks, collaboration networks, to terrorist networks, spread of epidemic diseases, dating sites, and so on. The textbook by Easley and Kleinberg [EK10] contains many more examples of social networks and their applications.

When sampling data from the real world, we are bound to get data that deviates from the intended model. This can be false data due to faulty, undersensitive or oversensitive equipment, due to incorrectly filled forms, due to software malfunctions, errors during communication transmission etc. In other cases, the data is simply not perfect, and might be contaminated for other reasons. However, if we have reasons to believe that such errors are few, we can still reason about the underlying platonic model, provided we can compute which samples are most likely to be flawed. As above in the dinner party example, we may come up with such an ideal platonic model of how we would like or expect a network to look and behave. Above, we expected, or at least wanted the network to look like that of a

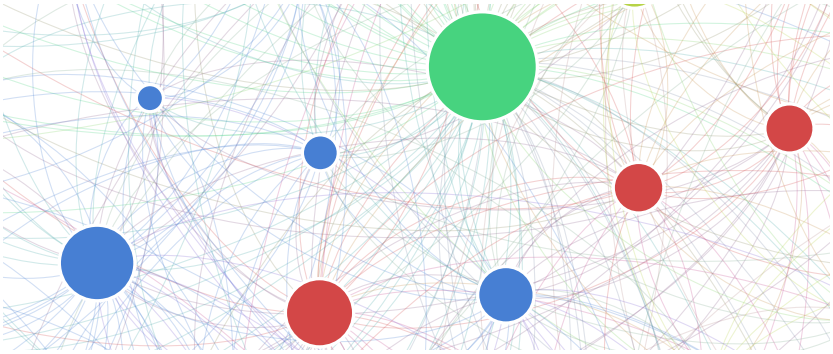
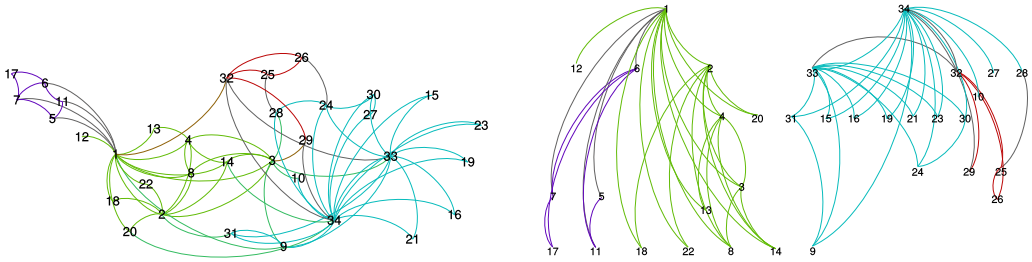


Figure 1.4: The entanglement of a small social network

cluster graph. But this is only one example of a model. If we analyze an email correspondence network in a company, we might expect to see a completely different kind of structure, perhaps a graph with a hierarchical structure. According to Nastos and Gao [NG13], this type of networks might be what is called trivially perfect graphs. A trivially perfect graph (we study this graph in much more details later) is a graph that has a certain hierarchy structure, where links on cross of groups are "illegal", whereas all links from a level above have to be present to a level below. However, just like above, if we are given the communication network in a company, we would have to expect certain communication links being present that do not adhere to the hierarchy chain.

In the case where people across groups communicate, we have an example of a false positive—a link that should not have been present. Similarly, it is perhaps not expected that the boss emails all her subordinates, and this corresponds to false negatives—a link that should have been there, but is not present. Assuming there are few false positives and negatives, we can apply tools from graph modification theory to get to the core of the communication network, and thus reveal the correct structure. Nastos and Gao propose that the most natural tool to apply would be what we will refer to throughout this thesis as Trivially Perfect Editing—what is the underlying hierarchical structure provided that some edges are missing and some edges are there that should not have been.

Yet another application of graph modification problems in social networks is that of finding so-called *central* nodes in social networks. Zachary's famous karate club analysis [Zac77] and Krebs analysis of terrorist networks [Kre02] are just two of many cases in the social sciences where the importance of nodes—or the centrality—has been studied from the network structure alone. This problem is called finding a *centrality measure* in a social network. However, there is not an agreed-upon concept of what does and does not make a central node. And until very recently, the only agreed-upon metric was that a node c was the most central among a group of nodes if c was the center of an *induced star* with the rest of the nodes being the leaves [Fre79]. Brandes argues that there is a very natural generalization of this concept using a *threshold graph* instead of a star, and that a node is more central than another node if it is of higher degree in an induced threshold graph [Bra14]. Brandes suggests that the right tool for the job is Threshold Editing.



(a) Zachary's karate club analysis. (b) An underlying hierarchy.

Figure 1.5: On the left, the data of Zachary's karate club analysis [Zac77]. When applying Trivially Perfect Editing to Zachary's data, we obtain a hierarchy model similar to that on the right [NG13]

The computational complexity of solving editing problems. As was hinted to above, it is actually not trivial to solve such instances optimally. And as the instances become larger, they become much harder. Ideally, we would like—or we would at least accept—that if a problem becomes twice as big, it takes twice the time to solve it. In the above case, we had 15 guests, and we would accept that in a dinner party with 30 guests, we used twice as much time to make the seating arrangement. Unfortunately, that is not how computational complexity, and indeed mathematics works. In some of these problems, if we add *just one more guest*, it would take twice the time. If we add ten new guests, it takes us thousand times the time it would take us to solve the original instance—as far as we know! And if this doesn't warrant further study—well—then what does?

The field concerned with these types of questions is the field algorithmic graph theory. In this field, algorithms for graph problems are designed, and the computational complexity of problems classified. Before we start constructing algorithms, or as a part of constructing algorithms, we determine the overall complexity of the problem. As discussed above, there are the types of problems that scale nicely; twice the dinner party, twice the computational time to solve. We will denote these problems by P.[2] And then there is the other type of problems, the ones where adding a few guests really blows up the computing time, and thus drains the battery on the laptop used to prepare the party. For the remainder of this introduction, we take P to mean *good*, and NPc to mean bad.

Unfortunately, it turned out that classifying the complexity of these editing problems was not as simple as it sounded. While for other types of problems, like vertex deletion problems that had the complexity classified for almost all of the natural problems immediately in 1980 [LY80], we still today are very far away from general knowledge about the complexity of edge editing problems—and that is not for lack of trying. We will spend the rest of this section highlighting the long and weary path that editing problems have taken, and I hope to convince

[2]The problems we denote by P will actually be all the problems for which there exists a polynomial time algorithm, hence the P. It is not only the problem names that are written in a special way, we typeset complexity classes in *sans-serif*.

the reader that despite all the work gone into classifying the complexity of these problems, they remain a mystery.

One of the first studied editing problems was Split Editing. The definition of split graphs is beyond the scope of this introduction, but is explained in later chapters. Perhaps it wasn't surprising back then, but in hindsight it could easily have been. Hammer and Simeone [HS81] showed that this problem is a P problem. That is a very good and promising start.

Five years later, Křivánek and Morávek, in 1986, showed that Cluster Editing, the one we used for the seating arrangement above, was one of the problems labelled NPc [KM86]. The problem was first proposed in 1964 [Zah64], so it took more than 20 years to really understand that this problem was a fundamentally difficult one.

Ten years passed until the next editing problem was classified. Studying so-called "physical mapping of DNA", Cirino, Muthukrishnan, Narayanaswamy, and Ramesh suggested using a model called *bipartite interval graphs*. Imagine here, if you will, thousands of DNA strands with partially overlapping endpoints, aligned on a line. They showed that there is little hope of computing these models efficiently; Bipartite Interval Editing was shown to be in NPc [CMNR97]. They also offer this nice intuition for why editing is hard:

> While edition using insertions and deletions separately has been well studied for various classes of graphs, graph edition problems appear harder when both addition and deletion operations are allowed since they can take potentially many more paths from the source graph to the target graph passing through intermediate graphs which have little, or nothing, in common with both.
> — Cirino et al. [CMNR97]

After year 2000, people started studying these problems more systematically in an attempt to obtain more general results and a better theory behind editing in general. Natanzon, Shamir, and Sharan [NSS01] showed that Perfect Editing, and Comparability Editing were NPc. Shamir, Sharan, and Tsur refined the result from above that the dinner party problem was NPc: They showed that even if you know in advance exactly how many tables you want, the problem is NPc unless you put everyone at the same table. In the language of mathematics, the problem ℓ-Cluster Editing is NPc for $\ell \geq 2$ and P otherwise [SST04].

Then, in 2006, Burzyn, Bonomo, and Durán managed to prove a wide array of editing problems to be NPc [BBD06], including editing to (unit) (circular) interval/arc graphs, permutation graphs and circle graphs. Still no sign of another problem labelled P. They conclude by posing nine open questions, seven of which on the complexity of editing problems.

In 2009 Alon and Stav [AS09] gave another general theorem for editing problems; If we, for some ℓ want to edit away all the cycles in the graph on exactly ℓ vertices, we are out of luck: C_ℓ Editing is NPc for every $\ell \geq 4$. When $\ell = 3$, the

problem is equivalent to the deletion version which was shown to be NPc by Yannakakis [Yan81b]. For $\ell \leq 2$ the problem does not make much sense.

The computational complexity of Cograph Editing one of the questions asked by Natanzon et al. in 2001 and then again by Burzyn et al. in 2006, but was not shown to be NPc until Liu, Wang, Guo, and Chen just a few years ago [LWGC12].

Trivially Perfect Editing was asked by Burzyn et al. in 2006, and was shown to be NPc independently by Nastos and Gao [NG13], and Drange and Pilipczuk [DP15]. Recall from above that Nastos and Gao studied this problem in a social networks setting, arguing that the perfect model for hierarchies within social networks are exactly the graphs trivially perfect graphs model (see Figure 1.5).

Chordal Editing was announced to be NPc by several sources [Nat99, NSS01, Sha02], however, all of these articles cite private communication with Ben-Dor. To the best of the author's knowledge, no proof was published until Drange, Dregi, Lokshtanov, and Sullivan [DDLS15] published a proof, while showing that Threshold Editing as well as Chain Editing are NPc, resolving a conjecture from Natanzon et al. [NSS01]. Recall also that Threshold Editing was the problem suggested by Brandes to be used for measuring the centrality of a node in a network.

1.2 Theoretical framework and related work

We now slowly move into the more theoretical and technical parts of this thesis. However, we will very quickly move through the history of graph modification again, now with less focus on the editing problems, but rather on graph modification in general. Some of the fundamental problems in algorithmic graph theory are the different notions of graph modification. In a graph modification problem, we are given a graph target, specified by some property Π, and asked to modify a given input graph to achieve the property. Some of the most commonly studied properties are various connectivity constraints, covering and packing constraints and where Π is a graph class of interest. Examples of the latter are deleting edges to obtain an acyclic graph, to obtain a bipartite graph, deleting or adding edges—as seen above—to obtain cluster graphs, or adding edges to obtain a chordal graph. Each of these examples have important practical and theoretical applications. As far as allowed modifications are concerned, the most popular and well-behaved variant is that of the so-called vertex deletion problems. In a vertex deletion problem, we have in mind some class of graphs $\mathcal{G}$ and we are asked, given a graph G, what is the least number of vertices you have to delete in order to obtain a graph in $\mathcal{G}$. We refer to this number as k and call it the *budget* of our instance.

Krishnamoorthy and Deo were among the first to systematically study the computational complexity of these problems [KD79], showing the NP-completeness of 17 different problems, including $\mathcal{G}$ being the class of complete graphs (Clique), the edgeless graphs (Independent Set), the class of forests (Feedback Vertex

Set), the class of planar graphs (Planar Vertex Deletion), the class of graphs omitting a specific cycle (C_k-free Vertex Deletion), the class of chordal (Chordal Vertex Deletion), interval graphs (Interval Vertex Deletion), and many more.[3] We here say that a graph G is H-free if G does not contain H as an induced subgraph, or in other words, we can not delete vertices of G to obtain a graph isomorphic to H.

Following this result, Lewis and Yannakakis [LY80] published a generalized result which included many of the 17 results of Krishnamoorthy and Deo, when they showed that for all non-trivial *hereditary* graph classes $\mathcal{G}$, for which membership is polynomial time computable, the $\mathcal{G}$ Vertex Deletion problem is NP-complete. Here, a property Π is non-trivial if it is true for infinitely many graphs and false for infinitely many graphs. This requirement is necessary, as otherwise the problem is solvable in polynomial time. A graph class $\mathcal{G}$ is hereditary if for every $G \in \mathcal{G}$, if G' is an induced subgraph of G, then $G' \in \mathcal{G}$. In other words, $\mathcal{G}$ is closed under deleting vertices.

A natural question following these results was the question of obtaining similar dichotomies or strong theorems for the edge deletion problems. In an edge deletion problem, we have a fixed graph class $\mathcal{G}$ in mind, and we are asked for the least number of edges to remove from a given graph G to obtain a subgraph $G' \in \mathcal{G}$. We call these kind of problems simply $\mathcal{G}$ Deletion, when $\mathcal{G}$ is the target graph class. But why only consider deleting edges? In the Connectivity Augmentation Problem we are asked to modify the graph to increase its connectivity. Clearly, by deleting edges, we never increase the connectivity. This gives rise to the variant of the edge modification problems called completion problems; What is the least number of edges to add to G to obtain a supergraph $G^+ \in \mathcal{G}$? We refer to these problems as $\mathcal{G}$ Completion when $\mathcal{G}$ is the target graph class. It should here be noted that the deletion and the completion problems are not fundamentally different, as the deletion problem on G to a hereditary class $\mathcal{G}$ is equivalent to the completion problem on $\overline{G}$ to co-$\mathcal{G}$. That is, by looking at the complement graph and the complement property, we can swap deletion for completion, and vice versa.

All versions, deletion of vertices, deletion of edges as well as adding edges have practical applications. The Planar Deletion has obvious applications in layout optimization [Ull84, BL84], as the number describes how many "bridges" a circuit board needs, and was early shown to be NP-complete [LG77]. Cluster Vertex Deletion has applications in correlation clustering, the problem where we want to classify objects into natural clusters [HKMN10]. Minimum Fill-In—the problem of adding as few edges as possible to obtain a chordal graph—has applications in sparse matrix multiplications [Ros72, Tar75] and more. In this thesis we will refer to this problem as Chordal Completion since that highlights its relation to the other problems studied herein.

As we discussed in the first section, there is a third option when it comes to

[3] The reader is invited at this point to read the very accessible letter on the status of the P versus NP problem by Fortnow [For09].

edge modification problems. What if we allow both deletion and completion? The computational editing problems turn out to have many natural applications, one important example being the correlation clustering problem [BBC04]. In this problem we are given a graph, and asked to add and delete as few edges as possible to obtain a *cluster graph*, that is, a graph whose connected components induce complete graphs.

Unfortunately, the way vertex deletion problems behaves nicely as seen above, with respect to NP-hardness, is exactly the way edge deletion problems do not. We will now quickly go through some of the goals, obstacles, and achievements in edge modification problems the last forty years.

> *It* [...] *would be nice if the same kind of techniques could be applied to the edge-deletion problems. Unfortunately we suspect that this is not the case — the reductions we found for the properties considered* [...] *do not seem to fall into a pattern.*
> — Yannakakis [Yan81b].

Garey, Gavril and Johnson proved in 1977 [GJ79] that Interval Completion was NP-complete. Following this result, using ad-hoc reductions, Yannakakis [Yan81a] showed Chain Completion and Chordal Completion to be NP-complete, first conjectured to be NP-hard by Rose, Tarjan, and Lueker [RTL76, RT78, Yan81a]. One of the first results towards a more general theorem was given by Watanabe, Ae, and Nakamura [WAN81] who showed that all edge deletion towards "finitely characterizable by 3-connected graph" problems were NP-complete.

Goldberg, Golumbic, Kaplan, and Shamir [GGKS95] noted, in a study originating from problems in molecular and computational biology, that from Chordal Completion [Yan81a], Interval Completion as well as Unit Interval Completion are NP-complete. They considered four simplified versions of physical mapping of DNA, the problem of "reconstructing the relative position of fragments of DNA along the genome from information on their pairwise overlaps" [GGKS95], finding that most of these problems are NP-hard.

Towards a more classification type of result, El-Mallah and Colbourn [EC88] showed that P_ℓ-free Deletion is NP-complete, where P_ℓ is the path on ℓ vertices, if and only if $\ell \geq 3$. For $\ell = 3$, this is exactly Cluster Deletion, and for $\ell = 4$, this is the Cograph Deletion problem. Reducing from the Clique problem, Margot showed that Threshold Completion is NP-complete [Mar94], where threshold graphs are the cographs that do not contain C_4 nor $\overline{C_4}$ as induced subgraphs. The graph C_4 is here the cycle on four vertices. See Table 2.1 for the author's private collection of small graphs.

Graph completion. The $\mathcal{H}$-free Completion problems form a subclass of *graph modification problems* where one is asked to add a bounded number of edges to an input graph to obtain a graph which is $\mathcal{H}$-free, where $\mathcal{H}$ is a set of forbidden induced subgraphs. We note here that the completion problems are not fundamentally different from deletion problems, especially not in $\mathcal{H}$-free graphs,

since the problem $\mathcal{H}$ Completion is polynomially equivalent to $\mathcal{H}'$ Deletion, where $\mathcal{H}'$ is the pointwise complement of $\mathcal{H}$.

One of the motivations to study completion problems in graph algorithms comes from their intimate connections to different *width parameters*. For example, the treewidth of a graph, one of the most fundamental *graph parameters*, is the minimum over all possible completions into a chordal graph of the maximum clique size minus one [Bod98]. The treedepth of a graph, also known as the vertex ranking number, the ordered chromatic number, and the minimum elimination tree height, plays a crucial role in the theory of sparse graphs developed by Nešetřil and Ossona de Mendez [NOdM12]. Mirroring the connection between treewidth and chordal graphs, the treedepth of a graph can be defined as the largest clique size in a completion to a *trivially perfect graph*. Similarly, the vertex cover number of a graph is equal to the minimum of the largest clique size taken over all completions to a *threshold graph*, minus one.

Interval Completion and Proper Interval Completion have strong connections to width parameters just like the ones mentioned above: The *pathwidth* of a graph is the minimum over the maximum clique size in an *interval completion* of the graph, minus one, whereas the *bandwidth* mirrors this relation for *proper interval completions* of the graph.

Much work has gone into the area of turning a graph chordal, one way or another, so much in fact as to have its own survey. Heggernes [Heg06] wrote a survey on *triangulation*, the task of adding edges to a graph to obtain a chordal graph, and note there the intricate relationship between these tasks and the task of computing the treewidth of a graph, sparse matrix computation [Ros72, Tar75], database management [BFMY83, TY84], and more.

1.3 Parameterized tractability

There are only few problems above that have polynomial time algorithms. They are relatively fast, which means we are able to solve relatively large instances relatively fast. These are the ones we like. However, for the NP-complete problems, no such algorithm can exist unless $\mathsf{P} = \mathsf{NP}$, which is widely believed to not be the case. So should we just give up? We still would like to be able to solve these problems, so what can we do? Dealing with the inherent intractability of many computational problems, like that of the problems mentioned above, is one of the main areas of modern algorithm research. To cope with the intractability, we are presented with a few options. We could either turn to approximation algorithms [WS11], where we are trying, not to find an optimal solution, but a solution that is good enough. Or we could accept the fate of exponential time, but try finding improved exact exponential time algorithms [FK10]. A completely different approach is to restrict our attention to smaller classes of input graphs [KS99, TNS82, GKLT09, CMPP14]. These are all classical ways of dealing with NP-complete problems.

A different perspective is offered by the field of *parameterized algorithms* and multivariate analysis. In all problems mentioned above, and in most naturally

occurring problems, we are interested in finding the *smallest* possible solution—we are looking for a solution of size at most some prescribed number k. In *parameterized complexity*, we are taking this value into account in the analysis of the running time. We are here looking for algorithms that solve problems in time $f(k) \cdot \text{poly}(n)$, where f can be any computable function with input k, called the *parameter*, and n is the size of the input, usually measured in the number of vertices in the input graph. However, poly is restricted to be a fixed polynomial function. A problem admitting such an algorithm is said to be *fixed-parameter tractable*. This means, informally and vaguely, that for "small enough" solutions, the problem is in some sense still tractable.

In some contrast to the chaos we have for edge modification problems with respect to their P vs. NP classification, much can be said about their parameterized complexity. Parameterized complexity offers a more fine-grained analysis than the P vs. NP classification does. As noted above, Yannakakis showed that for any polynomial time recognizable, hereditary non-trivial graph class, the vertex deletion problem to that graph class was NP-complete. This led Heggernes, Paul, Telle, and Villanger to ask if similar requirements were sufficient for obtaining fixed-parameter tractable algorithms [HPTV07]. However, not long after, Lokshtanov proved that there are problems, like Wheel-free Deletion that are W[2]-hard [Lok08], and thus unlikely to admit a fixed-parameter tractable algorithm. The theory of the W-hierarchy is beyond the scope of this thesis, but by "unlikely", we simply mean that if they do admit a fixed-parameter tractable algorithm, then the W-hierarchy collapses at the second level, i.e., $\mathsf{FPT} = \mathsf{W[1]} = \mathsf{W[2]}$, and this in turn implies that the exponential time hypothesis fails [LMS11]. We will discuss this hypothesis in more details later. Indeed, under the same hypothesis, we cannot even have an $f(k)n^{o(k)}$ algorithm for such problems.

However, we can restrict our attention to "finitely generated hereditary graphs", graphs *characterized by a finite set of forbidden induced subgraphs*. These are the graph classes $\mathcal{G}$ for which there is a finite list of graphs $H_1, H_2, \ldots, H_p$ such that any graph G belongs to $\mathcal{G}$ if and only if G does not contain any H_i as an induced subgraph. And there are many such graph classes. Indeed, most studied graph classes in this thesis are precisely such; cluster and bicluster graphs, threshold and chain graphs, trivially perfect and cographs, split and pseudosplit graphs, and many more. Cai [Cai96] discovered that for all graph modification problems towards these graph classes, there is a branching algorithm running in time $O(c^k n^c)$ for some constant c. Here, c depends only on the finite set of forbidden induced subgraphs. Although many studied graph classes satisfy this property, there are important examples, like chordal or interval graphs, that are outside this regime. This result does not directly cover problems like Chordal Completion, since chordal graphs cannot be characterized by a finite set of forbidden induced subgraphs. However, given an input instance (G, k) to Chordal Completion, we may observe that if G has an induced cycle of length more than $k+3$, then (G, k) is a no-instance [Cai96]. Hence, when k is the parameter, (G, k) is a yes instance of Chordal Completion if and only if (G, k) is a yes instance of $\mathcal{H}$-free

Completion for

$$\mathcal{H} = \{C_4, C_5, \ldots, C_{k+4}\},$$

and the output instance is chordal.

The optimality programme. Once a problem has been shown to admit a fixed-parameter tractable algorithm, a natural next question is whether it is possible to improve upon that algorithm. This is especially interesting when the algorithms have nasty running times like $2^{O(k^2)} \cdot \text{poly}(n)$, or even $2^{O(2^k)} \cdot \text{poly}(n)$. As mentioned above, the modification problems for finite forbidden induced subgraphs already have nice running times like $6^k \cdot \text{poly}(n)$ or even $3^k \cdot \text{poly}(n)$. Is it possible to obtain faster algorithms than what Cai's theorem provides? Can we improve from $3^k \,\text{poly}(n)$ to, say, $2^k \,\text{poly}(n)$ or $1.5^k \,\text{poly}(n)$?

> Clearly, it would be very useful to know if the current best algorithm can be improved further or it has already hit some fundamental barrier.
> — Marx [Mar12].

As Marx points out, it would be interesting to see if there are reasons to suspect that we cannot get better than $2^k \cdot \text{poly}(n)$ algorithms. We will return to that question later and indeed throughout this thesis, but first we take a detour into the world of compression and preprocessing.

1.4 Polynomial kernels

What can be done in polynomial time? An interesting question when a problem is NP-complete is: provided that we are only given polynomial time, what can we actually achieve within this running time? In this area, parameterized complexity offers a very intriguing concept: compression with mathematically grounded guarantees. Suppose that a problem is NP-complete, and let (G, k) be an input instance of this problem. Is it possible to, in polynomial time, compress the input instance (G, k) to an *equivalent* instance (G', k'), where the size of G' and k' are both bounded by $f(k)$ for some function f? We call an algorithm compressing the instance a *kernelization algorithm*. It turns out that when f is allowed to be any computable function, this question is equivalent to the question whether the problem is fixed-parameter tractable [DFS99]. But what happens if we require f to be a polynomial?

For vertex deletion problems the answer is again quite simple: As long as $\mathcal{G}$ is characterized by a finite set of forbidden induced subgraphs, the task is to hit all the copies of these subgraphs (so-called *obstacles*) that are originally contained in the graph. Hence, one can construct a simple reduction to the d-Hitting Set problem for a constant d depending on $\mathcal{G}$, and use the classic $O(k^d)$ kernel for the latter that is based on the sunflower lemma [FG06, AK10]. For edge modifications problems, however, this approach fails utterly: every edge addition and deletion can create new obstacles, and thus it is not sufficient to hit only the original ones.

For this reason, edge modification problems behave counter-intuitively with respect to polynomial kernelization, and up to recently very little was known about their complexity.

Kernelization of edge modification problems to classes that are characterized by a finite set of forbidden induced subgraphs has an interesting history. Already in 1999, Kaplan, Shamir, and Tarjan [KST99] showed that Chordal Completion admits a polynomial kernel with $O(k^5)$ vertices. This was later improved by Natanzon, Shamir, and Sharan [NSS00]. As above, chordal graphs do not have a finite set of forbidden induced subgraphs, but the set can be bounded by a function of k: if the graph contains an induced cycle of length more than $k+3$, then we can immediately infer that (G, k) is a no-instance. Gramm, Guo, Hüffner, and Niedermeier [GGHN08], and Guo [Guo07] showed kernels for several graph modification problems towards graph classes characterized by a finite set of forbidden induced subgraphs, including cluster, split, threshold, chain and trivially perfect graphs. Several positive results followed, which led Fellows, Langston, Rosamond, and Shaw to ask whether all $\mathcal{H}$-free modification problems admit polynomial, and even linear kernels [FLRS07].

This was refuted by Kratsch and Wahlström [KW13] using the framework of Bodlaender, Downey, Fellows, and Hermelin [BDFH09], who showed that for a certain graph on seven vertices, H_{KW} (see Table 2.1) with $\mathcal{H} = \{H_{\text{KW}}\}$, none of the problems $\mathcal{H}$-free Deletion or $\mathcal{H}$-free Editing, admit polynomial kernels unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.[4] This shows that the subtle differences between edge modification and vertex deletion problems have tremendous impact on the kernelization complexity. They conclude by asking whether there is a "simple" graph, like a path or a cycle, for which an edge modification problem does not admit a polynomial kernel under similar assumptions. This question was answered by Guillemot, Havet, Paul, and Perez [GHPP13] who showed that both for the class of P_ℓ-free graphs (for $\ell \geq 7$) and for the class of C_ℓ-free graphs (for $\ell \geq 4$), the edge deletion problems do not have polynomial kernelization algorithms, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. They simultaneously gave a cubic kernel for the Cograph Editing problem, the problem of editing to a graph without induced paths on four vertices.

These results were later improved by Cai and Cai [CC15], who attempted to obtain a complete dichotomy of the kernelization complexity of edge modification problems for classes of H-free graphs, for every graph H. The project has been very successful—the question is settled for all 3-connected graphs H, all paths and cycles, as well as all but a finite number of trees. In particular, it turns out that the existence of a polynomial kernel for any of H-Free Editing, H-Free Edge Deletion, or H-Free Completion problem is in fact a very rare phenomenon, and basically happens only for specific graphs H. For instance, for H being a path or a cycle, the aforementioned three problems admit polynomial kernels if and only if H has at most three edges.

[4] $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ implies that PH is contained in Σ_3^p. It is widely believed that PH does not collapse, and hence it is also believed that $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$.

There are several important and interesting open problems in the kernelization complexity of edge modification problems. To mention a few, Interval Completion and claw-free Deletion, and in addition a deterministic kernel for Edge Bipartization.

Towards these goals, some work has been done. Interval Completion was shown to be fixed-parameter tractable by Villanger, Heggernes, Paul, and Telle [VHPT09], where they conclude by asking specifically for a polynomial kernel. This question was raised again in the work of Bliznets, Fomin, Pilipczuk, and Pilipczuk [BFPP16]. Bessy and Perez [BP13] gave a polynomial kernel for Proper Interval Completion. Cygan et al. showed that deletion to a subclass of a claw-free graphs, diamond-claw-free Deletion admits a polynomial kernel [CPP+15], pinpointing the really hard cases of Claw-free Deletion one has to overcome before being able to obtain a polynomial kernel, whereas Cai showed that S_{11}-free Deletion does not have a kernel unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ [Cai12]. Here, S_{11} is the star on 11 vertices, and the claw is the star on four vertices. Kratsch and Wahlström [KW14] proved that there exists a randomized compression such that Edge Bipartization as well as the vertex version, Odd Cycle Transversal admits a $k^{4.5}$ co-RP kernel. Here, co-RP allows false positives in the sense that if an instance is a no-instance, then the compressed instance is a no-instance with probability at least 1/2. However, any yes-instance will be compressed to a yes-instance. Here, we may boost the success probability by running the algorithm polynomially in k many times (not polynomial in n as that would defeat the purpose of a kernelization procedure), and the output instance will then be the "and" over all the compressed instances.

For more on polynomial kernels with respect to the aforementioned graph classes, we may consult the recent survey on the kernelization complexity by Liu, Wang, and Guo [LWG14]. This survey is an investigative approach exploring the frontier of the kernelization complexity of graph modification problems.

1.5 Subexponential time algorithms

In the previous section, we focused on the compressibility of a problem. However, this is only a preprocessing procedure, and does not actually solve the problem. We return to the philosophy of the optimality programme. After having Cai's theorem showing that all the modification problems towards graph classes that are characterized by a finite set of forbidden induced subgraphs are solvable in fixed-parameter tractable time, a natural question to ask is about lower bounds. Cai's theorem says that completing to trivially perfect graphs can be done in $O(2^k n^5)$ time, so a natural question whether it is possible to improve that running time.

Simultaneously with the optimality programme and the development of polynomial kernel theory, there was a growing interest in identifying parameterized problems that are solvable in *subexponential parameterized time*, i.e., in time $2^{o(k)}\,\mathrm{poly}(n)$. Although for many classic parameterized problems already known NP-hardness reductions show that the existence of such an algorithm would contra-

dict the *exponential time hypothesis* of Impagliazzo et al. [IPZ01], subexponential parameterized algorithms were known to exist for problems in restricted settings, like planar, or more generally H-minor free graphs [DFHT05]. The complexity class of problems admitting such an algorithm is called SUBEPT and was defined by Flum and Grohe in their seminal work on parameterized complexity [FG06]. They noticed that most natural problems did, in fact, *not* live in this complexity class: The classical NP-hardness reductions paired with the *exponential time hypothesis* of Impagliazzo, Paturi, and Zane [IPZ01] is enough to show that no $2^{o(k)} \cdot \text{poly}(n)$ algorithm exists.

In this context, Chen posed the following question in the field of parameterized algorithms [BCC+06]: Are there examples of natural problems on graphs, that do not have such a topological constraint, and also have subexponential parameterized running time? As a reply to this question, Alon, Lokshtanov, and Saurabh [ALS09] devised an algorithm solving Feedback Arc Set on tournament graphs in time $2^{O(\sqrt{k}\log k)} \cdot \text{poly}(n)$. However, the aforementioned graph classes with topological constraints are sparse, and tournament graphs[5] are extremely dense. Hence, Chen's question was therefore not fully answered—are there problems which are in SUBEPT on general graphs?

The question was settled in full when Fomin and Villanger [FV13] gave an algorithm for Chordal Completion. Numerous $2^{O(k)}\,\text{poly}(n)$ algorithms were known [Cai96, KST99, BHV11] for this problem, but the surprising part here was that Fomin and Villanger proved that this problem was solvable in time $2^{O(\sqrt{k}\log k)} + \text{poly}(n)$. The additive polynomial factor was due to first preprocessing the graph, thereby obtaining a kernelized instance of polynomial size. The main tool in this algorithm was that of *minimal triangulations and potential maximal cliques*, a framework constructed earlier by Bouchitté and Todinca [BT01, BT02] applying tools both from there, and from the work by Fomin, Kratsch, Todinca, and Villanger on exact algorithms for Treewidth and Minimum Fill-In [FKTV08].

Following the results of Fomin and Villanger, several new subexponential parameterized time completion results followed. Based on the aforementioned work by Alon et al. [ALS09], Ghosh et al. [GKK+15] gave an algorithm with the same running time, $2^{O(\sqrt{k}\log k)} + \text{poly}(n)$, for Split Completion, thus also giving an algorithm for the equivalent problem of *deleting* to a split graph. A natural question arose again on the complexity of completing to $\mathcal{H}$-free graphs: Could this be subexponential time for all $\mathcal{H}$. For connected? While the classes of chordal and split graphs are rather "simple", they certainly are much more complex than the simple cluster or bicluster graphs. Therefore, the problems Cluster Editing and Cluster Deletion were natural candidates for subexponential time algorithms, together with the similar question for bicluster graphs.

This question was first answered in the negative by Komusiewicz and Uhlmann studying this problem on bounded degree graphs [KU12], and then independently by Fomin et al. [FKP+14]. Surprisingly, we cannot expect that such algorithms

[5] A directed graph is a tournament if between every pair of vertices u and v, either uv is an arc, or vu is an arc. They can be formed by taking a complete graph and directing all the edges.

exist. Komusiewicz and Uhlmann gave an elegant reduction proving that both parameterized and exact subexponential time algorithms were not achievable, unless the exponential time hypothesis fails [KU12].

Following the subexponentiality results of Chordal Completion and Split Completion, it was shown that Trivially Perfect Completion, as well as Chain Completion, Threshold Completion, and Pseudosplit Completion all were solvable in subexponential parameterized time [DFPV14]. Then followed two results by Bliznets et al. [BFPP16, BFPP14], that Interval Completion and Proper Interval Completion both are solvable in subexponential time, $2^{O(\sqrt{k}\log k)}\operatorname{poly}(n)$ and $2^{O(k^{2/3}\log k)}+\operatorname{poly}(n)$, respectively.

Later a problem known as Clique Editing, or Sparse Split Editing was introduced as a model for core/periphery structures [BE00], and for noise reduction [DM14a]. This problem consists of editing a graph to a disjoint union of a clique and an independent set, or, $\{2K_2, P_3\}$-free Editing. It should come as no surprise that the problem is solvable in subexponential time; A polynomial kernel is quite trivial after a twin reduction rule, and then the result follows from guessing a vertex in the clique and its (small) neighborhood difference.

Damaschke and Mogren show several similar problems to be solvable in subexponential parameterized and show that Clique Deletion is solvable in time $O^\star(1.6355^{\sqrt{k\ln k}})$ [DM14a]. Whether the Clique Editing problem was NP-hard, was asked as an open problem in IWOCA 2013, and answered independently by Damaschke and Mogren [DM14a] and Kovác, Selecéniová, and Steinová [KSS14].

Until recently, it was unknown whether Threshold Editing—as well as the very similar Chain Editing—was NP-hard or not. This was shown by Drange, Dregi, Lokshtanov, and Sullivan [DDLS15] where it was simultaneously given algorithms for both problems that run in time $2^{O(\sqrt{k}\log k)}+\operatorname{poly}(n)$, thereby adding these problems to the line of subexponential parameterized time solvable problems.

Targets with few components. As mentioned above, Cluster Editing is unlikely to admit a subexponential time algorithm [KU12]. On the other hand, the problem p-Cluster Editing, where the number of components in the target class is fixed to be at most p—rather surprisingly—does indeed admit a subexponential parameterized time algorithm; This was shown by Fomin et al. [FKP+14], who designed an algorithm solving this problem in time $2^{O(\sqrt{pk})}\cdot\operatorname{poly}(n)$. The p-Cluster Editing problem, as well as p-Cluster Deletion was first studied by Shamir, Sharan, and Tsur [SST04], who showed that even 2-Cluster Deletion was NP-complete, and furthermore that Cluster Deletion was NP-hard to approximate within a constant factor.

The subexponential parameterized explanation by Fomin et al. [FKP+14] was rebutted by Misra, Panolan and Saurabh [MPS13], showing that this parameter does not always help in the aim of obtaining subexponential time algorithm. They showed that p-Club d-Cluster Editing does not admit a subexponential time fixed parameter algorithm when parameterized on solution size and the number of connected components in the target graph class. Even more problems

have been studied under this dual parameterization. Drange, Reidl, Sánchez Villaamil, and Sikdar [DRSVS15] considered the extension of p-Cluster Editing to Bicluster Editing and the more general t-Partite Cluster Editing, yielding the problems p-Bicluster Editing and t-Partite p-Cluster Editing. None of the classical parameterized versions are solvable in subexponential time, but fixing the number of connected components in the solution, p, the problems become solvable in subexponential time. There it is shown that a problem called p-Starforest Editing is solvable in time $O(2^{3\sqrt{pk}} + m + n)$, whereas an algorithm of running time $2^{O(p\sqrt{k}\log(pk))} + O(m+n)$ is given for Bicluster Editing.

Lower bounds. On the other side of the subexponential time parameterized complexity is the lower bounds aspect. Here we have two things to take into account, *(i)* is a problem solvable in time $2^{o(k)}\operatorname{poly}(n)$, and supposing it is: *(ii)* what is the optimal running time. Is Chordal Completion solvable in $2^{O(\sqrt{k})}\operatorname{poly}(n)$ time, or is the log factor essential in the exponent? Recall that Split Completion *is* solvable in time $2^{O(\sqrt{k})}\operatorname{poly}(n)$ [CFK+15].

For the first part *(i)*, as mentioned above, Cluster Editing is not solvable in subexponential time. Drange, Fomin, Pilipczuk, and Villanger [DFPV15] showed that also for Trivially Perfect Completion, or $\{C_4, P_4\}$-free Completion, as well as for Pseudosplit Completion and Threshold Completion, we have subexponential time algorithms. There it is also shown that for any subset $\mathcal{H}$ of $\{2K_2, C_4, P_4\}$, the problem $\mathcal{H}$-free Completion is solvable in subexponential time only when

$$\mathcal{H} = \{2K_2, C_4\},\ \{C_4, P_4\},\ \text{or}\ \{2K_2, C_4, P_4\}.$$

Even the simple problem Cograph Completion can not be solved in time subexponential in the solution, unless the exponential time hypothesis fails.

For the optimality programme *(ii)* for subexponential time algorithms, we do not have many strong results. Fomin and Villanger [FV13] noted that, unless the exponential time hypothesis fails, Chordal Completion cannot be solved in time $2^{o(k^{1/6})}\operatorname{poly}(n)$. However, in a recent article, Bliznets et al. [BCK+16] show that this can be tightened quite a bit: Unless the exponential time hypothesis fails, there is a positive natural number $c > 1$ such that Chordal Completion can not be solved in time $2^{O(k^{1/4}/\log^c k)}\operatorname{poly}(n)$, and the same lower bound result holds for Interval Completion, Proper Interval Completion, Trivially Perfect Completion, Threshold Completion, and Chain Completion. This, however, still leaves a gap for almost all the problems between $k^{1/2}$ and $k^{1/4}$ in the exponent. Is the correct running times for these problems closer to $2^{O(k^{1/4}/\log^c k)}\operatorname{poly}(n)$, to $2^{O(k^{1/2})} + \operatorname{poly}(n)$ or to $2^{O(\sqrt{k}\log k)} + \operatorname{poly}(n)$? The gap is slightly larger for Proper Interval Completion for which we only know an algorithm running in time $k^{O(k^{2/3})} + \operatorname{poly}(n)$ [BFPP14].

Under a stronger assumption, they show that none of the problems above can be solved in time $2^{o(\sqrt{k})}\operatorname{poly}(n)$. This stronger assumption is on the subexponential-time approximation scheme for Min Bisection on d-regular graphs. The formal

definitions behind this assumption is beyond the scope of this thesis, but generally speaking, the assumption says that we need an approximation algorithm for Min Bisection with factor arbitrarily close to one, and it needs to run in subexponential time. The current best *polynomial-time* approximation for Min Bisection is today on the order $\log \mathrm{OPT}$ [Räc08].

1.6 Organization and overview

1.6.1 Organization of thesis

This thesis is divided into three main parts. Part II contains results about the polynomial kernelizability of certain problems, Part III contains results on some problems with subexponential parameterized time running algorithms and finally in Part IV we give some evidence that certain problems are not solvable in subexponential time, and we also prove Threshold Editing, Chain Editing, Chordal Editing as well as Trivially Perfect Editing and Cograph Editing to be NP-complete.

The main problems studied are modification towards threshold graphs and trivially perfect graphs. In each of the parts we add results on specific problems related to these two graph classes, in addition to several other results. In Part II, we show polynomial kernels for all three modification problems related to threshold graphs and trivially perfect graphs (quadratic, and septic vertex kernels, respectively).

In Part III, we show that all three modification problems towards threshold graphs and chain graphs are solvable in subexponential time, and we show that so is the completion version towards trivially perfect graphs. All of these algorithms run in time $2^{O(\sqrt{k}\log k)} + \mathrm{poly}(n)$. To complement this, we show in Part IV that neither editing, nor deleting towards trivially perfect graphs can be done within this running time; Provided that the exponential time hypothesis holds, there cannot be any algorithm solving Trivially Perfect Editing, nor the deletion version, in time $2^{o(k)} \cdot \mathrm{poly}(n)$, nor in exact time $2^{o(n+m)}$. We also show that the subexponential time algorithm for Threshold Editing is worth studying, by showing that this problem, and Trivially Perfect Editing are NP-complete.

In the last part, Part V, we conclude with some open questions and possible future directions in this field.

	Problem		
Graph class	Deletion	Completion	Editing
Trivially Perfect	k^7 (Thm. 4)	same as deletion	k^7 (Thm. 3)
Threshold	same as editing	same as editing	k^2 (Thm. 1)
Chain	same as editing	same as editing	k^2 (Thm. 2)
t-partite p-cluster	same as editing	—	tpk (Thm. 5)
$\mathcal{H}$-free*	same as editing	No kernel (Thm. 7)	k^c (Thm. 6)

Table 1.1: Summary of the kernelization results of this thesis. The kernel sizes are to be interpreted as $O(\cdot)$. The row marked with a star (*) is on bounded degree input graphs, for finite $\mathcal{H}$, and c depends only on the degree bound and on $\mathcal{H}$. The negative result is under the assumption that $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$.

1.6.2 Summary of the results

Kernels. We give several new polynomial kernel results, some for which the problem was previously unknown to be NP-complete. We improve existing kernels for Threshold Completion thereby answering recent open questions. We also improve and extend the known existing kernels for $\mathcal{H}$-free Deletion on bounded degree graphs to also take into account $\mathcal{H}$ being possibly disconnected and to work for the editing version.

Threshold and chain graphs
Threshold Editing, Threshold Completion, and Threshold Deletion, as well as modifications towards chain graphs admit quadratic vertex kernels.

Trivially perfect graphs
Trivially Perfect Editing and Trivially Perfect Deletion admit $O(k^7)$ vertex kernels. The kernel works for the completion version, whose polynomial kernel complexity status was already announced in 2007 by Guo [Guo07].

Starforest and multipartite cluster graphs
We give an $O(tpk)$ vertex kernel for the general problem t-partite p-cluster Editing, implying an $O(pk)$ vertex kernel for p-Bicluster Editing.

On bounded degree input
Finally, on bounded degree input graphs, we obtain polynomial kernels for every editing and deletion problem towards $\mathcal{H}$-free graphs, provided only that $\mathcal{H}$ is finite. We complement this result by showing that the completion problem will not admit such procedures unless the polynomial hierarchy collapses.

	Problem		
Graph class	Deletion	Completion	Editing
Cograph	same as editing	same as editing	ETH (Thm. 20)
Trivially Perfect	ETH (Cor. 14.5)	$c^{\sqrt{k}\log k}$ (Thm. 10)	ETH (Thm. 19)
Threshold	$c^{\sqrt{k}\log k}$ (Thm. 8)	same as deletion	NPc (Thm. 16) $c^{\sqrt{k}\log k}$ (Cor. 9.14)
Chain	$c^{\sqrt{k}\log k}$ (Cor. 9.16)	same as deletion	NPc (Thm. 17) $c^{\sqrt{k}\log k}$ (Thm. 9)
Pseudosplit	$c^{\sqrt{k}\log k}$ (Thm. 12)	same as deletion	$O(n^8)$ (Thm. 11)
C_4-free	ETH (Thm. 21)	ETH (Thm. 22)	not considered
Bicluster	same as editing	—	ETH (Cor. 16.5)
p-Bicluster	same as editing	—	$c^{\sqrt{k}\log k}$ (Thm. 14)
Starforest	ETH (Cor. 16.4)	—	ETH (Thm. 23)
p-Starforest	same as editing	—	$2^{3\sqrt{pk}}$ (Thm. 13)

Table 1.2: Summary of the subexponentiality results of this thesis. All the running times should be taken as $O^\star(\,\cdot\,)$. ETH means that the problem is NP-complete, and that there is no $2^{o(k)}\,\mathrm{poly}(n)$ algorithm under the assumption of the exponential time hypothesis.

Subexponential time algorithms. We design several subexponential time algorithm, as well as a polynomial time algorithm for Pseudosplit Editing. Also here, some problems were not known to be NP-complete prior to this work. We describe algorithms solving all three modification problems towards threshold graphs and chain graphs, as well as completion towards trivially perfect graphs. These algorithms all run in time $2^{O(\sqrt{k}\log k)} + \mathrm{poly}(n)$.

For p-Bicluster Editing, we attempt to obtain similar running time as was done for p-Cluster Editing, but only achieve $2^{O(p\sqrt{k}\log(pk))} + \mathrm{poly}(n)$. For the special case of p-Starforest Editing, we did however achieve similar running times, namely $O(2^{3\sqrt{pk}} + n + m)$.

Threshold and chain graphs

Threshold Editing, Completion and Deletion, as well as modifications towards chain graphs, are all solvable in time $2^{O(\sqrt{k}\log k)} + \mathrm{poly}(n)$.

Trivially perfect graphs

Trivially Perfect Completion is solvable in time $2^{O(\sqrt{k}\log k)} + \mathrm{poly}(n)$.

Pseudosplit graphs

Pseudosplit Completion and its equivalent deletion variant are solvable

in time $2^{O(\sqrt{k}\log k)} + \text{poly}(n)$, and Pseudosplit Editing is solvable in polynomial time.

Starforest and multipartite cluster graphs

p-Starforest Editing is solvable in time $O(2^{3\sqrt{pk}} + n + m)$, and finally, both p-Bicluster Editing and t-partite p-cluster Editing admit algorithms on the form $2^{O(p\sqrt{k}\log(pk))} + \text{poly}(n)$, where p is the number of connected components in the target graph.

Lower bounds. In addition to the lower bound result that $\mathcal{H}$-free Completion does not admit a general polynomial kernelization procedure (unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$). We give numerous NP-completeness results and "ETH-hardness" results. The lower bounds of form $2^{o(\cdot)}$ below are all under the assumption of the validity of the exponential time hypothesis.

Threshold and chain graphs

Threshold Editing and Chain Editing are NP-complete. We massage the reduction into an NP-completeness result for Chordal Editing. The latter was known, but I have been unable to locate a proof for this in the literature.

Trivially perfect graphs

Trivially Perfect Editing is NP-complete. This was shown independently by Nastos and Gao [NG13], however, we also show that the problem does not admit a subexponential parameterized time algorithm unless the exponential time hypothesis, a result which cannot be deduced from their reduction as the reduction they provide suffers a cubic parameter blow-up. We give lower bounds on the form $2^{o(k)}\,\text{poly}(n)$ and even $2^{o(n+m)}$, also for Trivially Perfect Deletion.

Cographs graphs

Cograph Editing is shown to be NP-complete by a simpler and much smaller reduction than the original presented by Liu et al. [LWGC12]. Neither Cograph Editing nor Cograph Deletion admit subexponential time algorithms.

C_4-free graphs

Neither C_4-free Deletion nor C_4-free Completion admit subexponential time algorithms. They are both NP-complete.

Bicluster graphs and starforests

Finally, we show that Starforest Editing as well as the more general versions Bicluster Editing and t-partite p-cluster Editing are NP-complete on subcubic graphs and do not admit subexponential time algorithms, and if we parameterize by p alone, then p-Starforest Editing is W[1]-hard.

Chapter 2

Preliminaries

In this chapter we set up all the notation and basic definitions that are used throughout the thesis. We provide definitions, intuition and equivalent descriptions of all graph classes mentioned, and we describe the algorithmic toolbox and the tools of analysis we need. This entire chapter may readily be skipped and be read at the readers own discretion.

2.1 Graphs and modification problems

We consider only finite simple graphs $G = (V, E)$. We will denote by $V(G)$ and $E(G)$ the vertex set and edge set of a graph G, respectively, and we use n_G and m_G to denote their sizes. If A and B are sets of vertices, we write $E_G(A, B)$ to denote the set of edges of E with one endpoint in A and one in B. If $X \subseteq V(G)$ is a set of vertices, we denote by m_X the number of edges in X, i.e., $m_X = |E_G(X, X)|$. We denote by $N_G(v)$ the set of neighbors of v in G, and let $\deg_G(v) = |N_G(v)|$ denote the *degree* of v. If $N_G(v)$ is a clique, we say that v is *simplicial*. We denote by $\Delta(G)$ the *maximum degree* of G, that is,

$$\Delta(G) = \max_{v \in V(G)} \deg(v).$$

We say that a graph class $\mathcal{G}$ has bounded degree if there exists a constant $c \in \mathbb{N}$ such that $\max_{G \in \mathcal{G}} \Delta(G) \leq c$.

We omit subscripts when the graph in question is clear from context. We refer to the monograph by Diestel [Die05] for graph terminology and notation not defined here. For an introduction to parameterized complexity analysis, consult the monographs by Downey and Fellows [DF99, DF13], Flum and Grohe [FG06], and Niedermeier [Nie06]. For more on parameterized algorithms, see the textbook on parameterized algorithms by Cygan et al. [CFK+15].

We consider an edge in $E(G)$ to be a set of size two, i.e., $e \in E(G)$ is of the form $\{u, v\} \subseteq V(G)$ with $u \neq v$. We denote by $[V(G)]^2$ the set of all size two subsets of $V(G)$. When $F \subseteq [V(G)]^2$, we write $G \triangle F$ to denote $G' = (V, E \triangle F)$, where $\triangle$ is the *symmetric difference*, i.e., $E \triangle F = (E \setminus F) \cup (F \setminus E)$. When the graph is clear from context, we will refer to F simply as a set of edges rather

than $F \subseteq [V(G)]^2$. We will further write $G + F$ and $G - F$ for the graphs $(V(G), E(G) \cup F)$ and $(V(G), E(G) \setminus F)$, respectively.

For a graph G and a vertex v we define the *true twin class* of v, denoted $\mathrm{ttc}(v)$ as the set $\{u \in V(G) \mid N[u] = N[v]\}$. Similarly, we define the *false twin class* of v, denoted $\mathrm{ftc}(v)$ as the set $\{u \in V(G) \mid N(u) = N(v)\}$. Observe that either $\mathrm{ttc}(v) = \{v\}$ or $\mathrm{ftc}(v) = \{v\}$. From this we define the *twin class* of v, denoted $\mathrm{tc}(v)$ as $\mathrm{ttc}(v)$ if $|\mathrm{ttc}(v)| > |\mathrm{ftc}(v)|$ and $\mathrm{ftc}(v)$ otherwise. We take $f(n) = \mathrm{poly}(n)$ to mean $f(n) = n^{O(1)}$, i.e., that there exists a $c \in \mathbb{N}$ such that $f(n) = O(n^c)$.

Definition 2.1 (X-neighborhood)**.** Let G be a graph and $X \subseteq V(G)$. For a vertex $v \in V(G) \setminus X$, the *X-neighborhood* of v, denoted $N_{G,X}(v)$, is the set $N_G(v) \cap X$. The family of X-neighborhoods of G is the set $\{N_{G,X}(v) : v \in V(G) \setminus X\}$. When the graph G is clear from context, we simply write $N_X(v)$.

Definition 2.2 (Isomorphism)**.** An isomorphism between two graphs G_1 and G_2 is a bijective function $\varphi\colon V(G_1) \to V(G_2)$ such that $vu \in E(G_1)$ if and only if $\varphi(v)\varphi(u) \in E(G_2)$. If there exists an isomorphism between G_1 and G_2, we say that they are *isomorphic*, and denote it by $G_1 \cong G_2$.

Definition 2.3. Let G and H be two graphs. We say that G contains a copy of H, or simply put, G contains H if there is a set of vertices $V_H \subseteq V(G)$ such that $G[V_H] \cong H$.

Definition 2.4 ($\mathcal{H}$-free)**.** A graph G is $\mathcal{H}$-free if G does not contain any of the graphs $H \in \mathcal{H}$.

We will sometimes abuse notation and write H-free for $\{H\}$-free.

The *diameter* of a connected graph G, denoted $\mathrm{diam}(G)$, is defined as the number of edges in a longest shortest path of G, i.e.,

$$\mathrm{diam}(G) = \max_{u,v \in V(G)} \mathrm{dist}_G(u, v).$$

If G is disconnected, we define $\mathrm{diam}(G)$ to be $\max_C \mathrm{diam}(C)$, over all connected components C of G. For a graph G, a vertex $v \in V(G)$ and a set of vertices $X \subseteq V(G)$ we define the *distance from v to X*, denoted $\mathrm{dist}(v, X)$ as $\min_{u \in X} \mathrm{dist}(v, u)$. When provided with a non-negative integer r in addition, we define the *ball around X of radius r*, denoted $B(X, r)$, as the set $\{v \in V(G)$ such that $\mathrm{dist}(v, X) \leq r\}$.

An *obstruction set* $\mathcal{H}$ is a finite set of graphs. Given an obstruction set $\mathcal{H}$, a graph G and an induced subgraph H of G we say that H is an *obstruction* in G if H is isomorphic to some element of $\mathcal{H}$. If there is no obstruction H in G we say that G is $\mathcal{H}$-free. The size of the largest graph in $\mathcal{H}$ we denote by $n_{\mathcal{H}} = \max\{|V(H)|$ for $H \in \mathcal{H}\}$. In addition, we lift the notation of diameter to account for a finite set of graphs $\mathcal{H}$, denoted $\mathrm{diam}(\mathcal{H})$, being the maximum of $\mathrm{diam}\, G$ for $G \in \mathcal{H}$.

Given a graph G and an integer k the problem $\mathcal{H}$-free Deletion asks whether there is a set $F \subseteq E(G)$ with $|F| \leq k$ such that $G - F$ is $\mathcal{H}$-free. And similarly, $\mathcal{H}$-free Editing asks whether there is a set $F \subseteq E(G)$ with $|F| \leq k$ such that $G \triangle F$ is $\mathcal{H}$-free. We say that a set of edges F is an *$\mathcal{H}$-solution* if $G \triangle F$ is $\mathcal{H}$-free. When $\mathcal{H}$ is clear from context, we will refer to F simply as a solution. When the problem at hand is the deletion problem, we furthermore assume $F \subseteq E(G)$, and when the problem at hand is the completion problem, we assume $F \cap E(G) = \emptyset$, as is expected.

Definition 2.5 (H-packing)**.** Given a graph G and an obstruction H we say that $\mathcal{X}$ forms an *H-packing* in G if

(i) $G[X]$ and H are isomorphic for every $X \in \mathcal{X}$, and

(ii) X and Y are disjoint for every $X, Y \in \mathcal{X}$.

Observation 2.6. *Given a graph G and an obstruction H we can obtain a maximal H-packing $\mathcal{X}$ in $O(n^{|V(H)|+1})$ time.*

2.1.1 The three edge modification problems

We have already heard a lot about graph modification problems, and here we formally define the three general edge modification problems mentioned above. Below, we take $\mathcal{H}$ to be any set of simple graphs. Recall that a graph G is $\mathcal{H}$-free if for every obstruction $H \in \mathcal{H}$, the graph G does not contain H as an induced subgraph.

$\mathcal{H}$-free Completion parameterized by k

Input:	A graph G, and an integer k
Question:	Does there exist a set F of at most k edges such that $G + F$ is $\mathcal{H}$-free?

$\mathcal{H}$-free Deletion parameterized by k

Input:	A graph G, and an integer k
Question:	Does there exist a set F of at most k edges such that $G - F$ is $\mathcal{H}$-free?

$\mathcal{H}$-free Editing parameterized by k

Input:	A graph G, and an integer k
Question:	Does there exist a set F of at most k edges such that $G \triangle F$ is $\mathcal{H}$-free?

We say that the *pointwise complement* of $\mathcal{H} = \{H_1, H_2, \dots\}$, where $\mathcal{H}$ is any set of graphs, is the set $\overline{\mathcal{H}} = \bigcup_{H \in \mathcal{H}} \overline{H}$. Observe then that the class of $\mathcal{H}$-free graphs are closed under complements.

Fact 2.7. *If $\mathcal{H}$ is its own pointwise complement, then $\mathcal{H}$-free* Completion *is polynomial time equivalent to $\overline{\mathcal{H}}$-free* Deletion.

2.1.2 Modules and modular decomposition

In our kernelization algorithm for the modification problems towards trivially perfect graphs, we will apply the notion of a *module* in a graph.

Definition 2.8. Given a graph G, a set of vertices $M \subseteq V(G)$ is called a *module* if for any two vertices v and u in M, we have that $N(v) \setminus M = N(u) \setminus M$, i.e., all the vertices of M have exactly the same neighborhood outside M.

Observe that for any graph G, any singleton $M = \{v\}$ is a module, and also $V(G)$ itself is a module. However, G can contain a whole hierarchy of modules. This hierarchy can be captured using the following notion of a *modular decomposition*, introduced by Gallai [Gal67]. The following description of a modular decomposition is taken verbatim from the work of Bliznets et al. [BFPP16].

A module decomposition of a graph G is a rooted tree T, where each node t is labeled by a module $M^t \subseteq V(G)$, and is one of four types:

leaf
: t is a leaf of T, and M^t is a singleton;

union
: $G[M^t]$ is disconnected, and the children of t are labeled with different connected components of $G[M^t]$;

join
: the complement of $G[M^t]$ is disconnected, and the children of t are labeled with different connected components of the complement of $G[M^t]$;

prime
: neither of the above holds, and the children of t are labeled with different modules of G that are proper subsets of M^t, and are inclusion-wise maximal with this property.

Moreover, we require that the root of T is labeled with the module $V(G)$. We need the following properties of the module decomposition.

Proposition 2.9 ([MS99])**.** *For a graph G, the following holds.*

1. *A module decomposition $(T, (M^t)_{t \in V(T)})$ of G exists, is unique, and computable in linear time.*

2. *At any prime node t of T, the labels of the children form a partition of M^t. In particular, for each vertex v of G there exists exactly one leaf node with label $\{v\}$.*

3. *Each module M of G is either a label of some node of T, or there exists a* ***union*** *or* ***join*** *node t such that M is a union of labels of some children of t.*

Since in this work we do not aim at optimizing the running time of the kernelization algorithms, we do not need to compute the modular decomposition in linear time. Any simpler polynomial time algorithm would suffice (see the work of McConnell and Spinrad [MS99] for a literature overview).

Definition 2.10 (Laminar)**.** Given a set system $\mathcal{S} = (\mathcal{U}, \mathcal{F})$ over a universe $\mathcal{U}$, we say that $\mathcal{S}$ is *laminar* if for every two sets $F_1, F_2 \in \mathcal{F}$, either $F_1 \subseteq F_2$, $F_2 \subseteq F_1$ or $F_1 \cap F_2 = \emptyset$. In other words, every pair of sets in the system is either disjoint or nested. When the set system is clear from context, we will refer to $\mathcal{F}$ as laminar.

Lemma 2.11 (Folklore)**.** *Let $\mathcal{F}$ be a laminar set system over a finite ground set U. Then the cardinality of $\mathcal{F}$ is at most $2|U|$.*

Proof sketch. By associating the elements of U with the leaves of a rooted tree where each internal non-root node has degree at least three, a non-empty element of F corresponds exactly to a rooted induced subtree. Since this tree can have at most $2|U| - 1$ nodes, by adding the possibility of the empty set, we obtain the result. □

Definition 2.12 (Weakly laminar set system)**.** A set system $\mathcal{F} \subseteq 2^U$ over a ground set U is called a *weakly laminar set system* if for every X_1 and X_2 in $\mathcal{F}$ with $x_1 \in X_1 \setminus X_2$ and $x_2 \in X_2 \setminus X_1$, there is no $Y \in \mathcal{F}$ with $\{x_1, x_2\} \subseteq Y$.

The following property bounds the size of a weakly laminar set system, which we need later, and as it turns out, the size of a weakly laminar set system is even less than for a laminar set system:

Lemma 2.13. *Let $\mathcal{F}$ be a weakly laminar set system over a finite ground set U. Then the cardinality of $\mathcal{F}$ is at most $|U| + 1$.*

Proof. We proceed by induction on $|U|$, with the claim being trivial when $U = \emptyset$. Suppose $\mathcal{F}$ is a weakly laminar set system over a ground set U, and let X be a member of $\mathcal{F}$ that has the minimum cardinality among the nonempty ones (if there is no such set, then $|\mathcal{F}| \leq 1$ and we are done). The first observation is that if Y_1 and Y_2 are two nonempty members of $\mathcal{F}$ that satisfy $Y_1 \setminus X = Y_2 \setminus X$ (possibly $Y_1 = X$ or $Y_2 = X$), then in fact $Y_1 = Y_2$. Suppose otherwise that there exist two such nonempty sets $Y_1, Y_2 \in \mathcal{F}$ with $Y_1 \cap X \neq Y_2 \cap X$; Without loss of generality, suppose that there exists an element $x_1 \in Y_1 \setminus Y_2 \subseteq X$, and hence $x_1 \in X \setminus Y_2$.

Since X is of minimum cardinality, we have that $|X| \leq |Y_2|$. As $X \not\subseteq Y_2$, we infer that there exists an element $x_2 \in Y_2 \setminus X = Y_1 \setminus X$. Consider the pair $\{x_1, x_2\}$

and observe that *(a)* $x_1 \in X \setminus Y_2$, *(b)* $x_2 \in Y_2 \setminus X$, and *(c)* $\{x_1, x_2\} \subseteq Y_1$. This contradicts the definition of a weakly laminar set system.

Define a set system $\mathcal{F}'$ over the ground set $U \setminus X$ as follows:

$$\mathcal{F}' = \{Y \setminus X \,:\, Y \in \mathcal{F}, Y \neq \emptyset\}.$$

Clearly, $\mathcal{F}'$ is a weakly laminar set system over a strictly smaller ground set, so from the induction hypothesis we infer that $|\mathcal{F}'| \leq |U \setminus X| + 1$. Moreover, from the observation of the previous paragraph we infer that sets $Y \setminus X$ are pairwise different for $Y \in \mathcal{F}, Y \neq \emptyset$, and hence $|\mathcal{F}| \leq |\mathcal{F}'| + 1$ (the additive $+1$ comes from possibly having the empty set in $\mathcal{F}$). Concluding,

$$|\mathcal{F}| \leq |\mathcal{F}'| + 1 \leq |U \setminus X| + 1 + 1 \leq |U| - 1 + 1 + 1 = |U| + 1.$$

□

For our polynomial kernel hardness result, we reduce from the problem Cubic Planar Vertex Cover, which is the famous Vertex Cover problem restricted to regular planar input graphs of degree three. The following result shows the validity of reducing from this restricted instance:

Proposition 2.14 ([Moh01])**.** Vertex Cover *is* NP*-complete on cubic planar graphs.*

It is well known that planar graphs can be recognized in polynomial time, so an algorithm can simply reject the input if the graph is not regular or non-planar. When we later will make a cross-composition argument, we will reduce from Cubic Planar Vertex Cover, but the resulting budget will depend on the number of edges in the input graph!

Luckily, this is not a problem for us. We may allow us, on $t = 2^r$ instances of Cubic Planar Vertex Cover on n vertices, m edges and a budget of k, to have a budget of size bounded by $\text{polylog}(t) + \text{poly}(k) = \text{polylog}(t) + \text{poly}(n + m + k)$, as the following observation shows us:

Observation 2.15. *The vertex cover number of a cubic graph on n vertices is at least $m/3$. Since $m = n3/2$, $\text{vc}(G) \geq n/2$.*

2.1.3 Cheap or Expensive?

Given an instance (G, k) and a solution F, we define the *editing number* of a vertex v, denoted $\text{en}_G^F(v)$, to be the number of edges in F incident to a vertex v. When G and F are clear from the context, we will simply write $\text{en}(v)$. A vertex v will be referred to as *cheap* if $\text{en}(v) \leq 2\sqrt{k}$ and *expensive* otherwise. We will call a set of vertices $U \subseteq V$ *small* provided that $|U| \leq 2\sqrt{k}$ and *large* otherwise.

Definition 2.16. Given an instance (G, k) with solution F, we call a vertex v *cheap* if $\text{en}(v) \leq 2\sqrt{k}$.

The following observation will be used extensively.

Observation 2.17. *If $U \subseteq V(G)$ is a large set, then there exists a cheap vertex in U, or contrapositively: if a set $U \subseteq V(G)$ has only expensive vertices, then U is small. Specifically it follows that in any yes-instance (G, k) where F is a solution, there are at most $2\sqrt{k}$ expensive vertices.*

This gives the following win-win situation: If a set X is small, then we can "guess" it, in time $O(n^{2\sqrt{k}})$, which is subexponential provided that $n = \text{poly}(k)$. Hence we can in subexponential time enumerate all candidates, and otherwise, we can guess a cheap vertex inside the set and its "correct" neighborhood. In particular, since the set of expensive vertices is small, we can guess it in the beginning. We can always assume, with subexponential time overhead, that the graph G is a labeled graph, where some vertices are labeled as cheap and others as expensive. There will never be more than $2\sqrt{k}$ vertices labeled expensive, however a vertex labeled expensive might very well not be expensive in G and vice versa. The idea is then that we guess the expensive vertices at the start of the algorithm and then bring this information along when we recurse on subgraphs.

A crucial part of many of the subexponential time algorithms is to enumerate sets of size at most $O(\sqrt{k})$. The following lemma shows that as long as the instance of size polynomial in the parameter, this is indeed doable and we will use the result of this lemma throughout the thesis without necessarily referring to it.

Lemma 2.18. *For every $c \in \mathbb{N}$ there is an algorithm that, given an input instance (G, k) with $|V(G)| = \text{poly}(k)$ enumerates all vertex subsets of size $c\sqrt{k}$ in time $2^{O(\sqrt{k}\log k)}$.*

Proof. Given an input graph $G = (V, E)$ and a natural number k with $|V| = n = \text{poly}(k)$, we can simply output the family of sets $\mathcal{X} \subseteq 2^V$ of size at most $c\sqrt{k}$, which takes time

$$\sum_{\kappa \leq c\sqrt{k}} \binom{n}{\kappa} \leq c\sqrt{k}\binom{n}{c\sqrt{k}} \leq c\sqrt{k} \cdot n^{c\sqrt{k}} = 2^{O(\sqrt{k}\log n)} = 2^{O(\sqrt{k}\log k)},$$

where the first inequality follows since $\binom{n}{i}$ is increasing for i from 1 to $c\sqrt{k}$. □

2.2 Graph classes

In this chapter we recall some basic graph classes that will be important for understanding the results of this thesis. The majority of the results in the thesis concern *trivially perfect graphs* and *threshold graphs*. These graph classes are covered in detail below. However, being graph classes, they live in a vast jungle (see Figure 2.1[1]) of other graph classes, and it is important to understand the

[1]Note that the even-hole-free graphs are the graphs that are $\{C_6, C_8, C_{10}, C_{12}, \dots\}$-free, i.e. C_4 is not forbidden, as is used in the survey on graph classes [BLS99]; There, a *hole* is any cycle of length at least five.

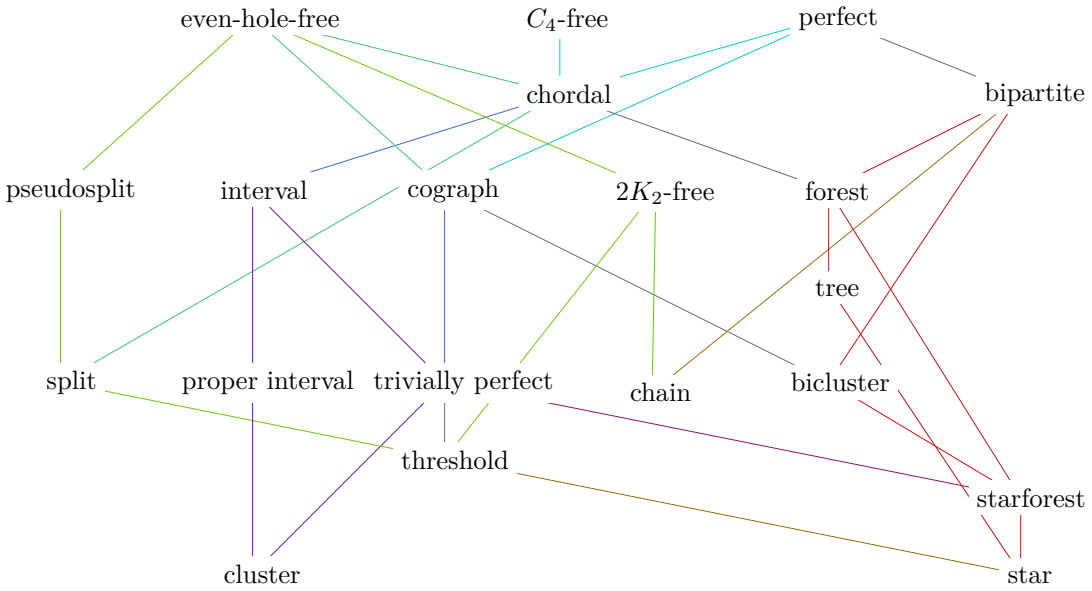


Figure 2.1: Jungle of graph classes we are concerned with in this thesis

nature of these graph classes, their relations with each other and their structure to fully appreciate the tools applied. The nice and helpful survey by Brandstädt, Le, and Spinrad on graph classes [BLS99] contains more than enough information on the graph classes studied here and can therefore readily be used as a reference.

2.2.1 Chordal and related graphs

The *chordal graphs* form a fundamental graph class. They are the graphs without induced cycles of length at least four, that is, they are precisely the $\{C_4, C_5, C_6, \dots\}$-free. More information on chordal graphs can be found in the survey on minimal triangulations by Heggernes [Heg06], which includes several important characterizations of chordal graphs and many pointers to the vast literature surrounding the study of chordal graphs. However, since we will not be working directly on chordal graphs, we will not cover this class in more details.

The *interval graphs* form a subclass of the chordal graphs. They are the *AT-free* chordal graphs; An *asteroidal triple*, or *AT* for short, of a graph G, is a triple of vertices $A = (v_1, v_2, v_3)$ such that for any vertex $x \in A$, the other two vertices are connected in $G - N[x]$. It is here understood that no two of these vertices can be adjacent. A graph is said to be AT-free if it does not contain an AT. Thus, being the *AT-free chordal graphs*, the interval graphs have a linear structure. There is also a geometric definition of this graph class. A graph $G = (V, E)$ is an interval graph if and only if there is a mapping φ from V to intervals on the real line such

that $vu \in E$ if and only if $\varphi(v)$ overlaps $\varphi(u)$. We call φ an *interval model* in the case where the graph determined by φ is isomorphic to G.

The *proper interval graphs*, also known as *unit interval graphs*, are exactly those interval graphs whose interval representation can be represented by intervals of unit length, hence the name unit interval graphs. An interval model can be represented using only unit length intervals if and only if it can be represented in a way such that no interval properly contains another interval, hence the name proper interval graphs. The proper interval graphs are exactly the *claw-free* interval graphs, where a claw is the star on four vertices.

Whereas chordal graphs were the graphs without cycles of length at least four, the *bipartite graphs* are exactly the graphs without odd cycles, i.e., it does not contain (neither induced nor as a subgraph) a cycle with an odd number of edges. It is an old folklore result that the following are equivalent:

- A graph is bipartite, i.e., its vertex set can be partitioned into two sets A and B such that for every edge uv, $|\{u, v\} \cap A| = 1$
- A graph does not contain an odd cycle as a subgraph
- A graph does not contain an odd cycle as an induced subgraph
- A graph is colorable with two colors such that no two adjacent vertices have the same color

We denote by $G = (A, B, E)$ a bipartite graph where $A \uplus B$ is the vertex set and $E \subseteq A \times B$. The complement of a bipartite graph is called a *cobipartite graph*.

2.2.2 Finite set of forbidden induced subgraphs

For a set of graphs $\mathcal{H}$, we say that G is *$\mathcal{H}$-free* if for every $H \in \mathcal{H}$, G does not have as an induced subgraph, an isomorphic copy of H. We denote by $\mathcal{G}_{\mathcal{H}}$ the graph class consisting of all graphs that are $\mathcal{H}$-free.

All the above graph classes can be characterized by such a set of *forbidden induced subgraphs*, i.e., for every graph class $\mathcal{G}$ above, there is a set $\mathcal{H}$ of graphs, such that $\mathcal{G} = \mathcal{G}_{\mathcal{H}}$ any graph G is in $\mathcal{G}$ if and only if G does not have any graph $H \in \mathcal{H}$ as an induced subgraph. This property is in fact equivalent to that of being a hereditary graph class.

There is, however, a special class of graph classes[2] that are characterized by a *finite set of forbidden induced subgraphs*. We say that a graph class $\mathcal{G}$ is characterized by a finite set of forbidden induced subgraphs if there is a finite set of graphs $\mathcal{H}$ such that $\mathcal{G} = \mathcal{G}_{\mathcal{H}}$.

[2]It is worthwhile to make clear that even though we talk about *graph classes* and *classes* of graph classes, we are in fact talking about sets. As long as we consider a graph to be represented by its "isomorphism class" we are in fact only dealing with $\aleph_0$ and $\aleph_1$ sized sets here; There is a countable number of graph classes characterized by finite sets of forbidden induced subgraphs, and $\aleph_1$ many hereditary graph classes.

Split and pseudosplit graphs. The class of split graphs is a fundamental and old graph class which has been studied intensely. It is a very simple graph class to define, but a graph class hard to master. Being exactly the intersection of chordal and co-chordal graphs [FH76], a graph G is a split graph if its vertex set can be partitioned into two sets C and I such that C is a clique and I is an independent set. We call (C, I) a *split partition* of G. Observe that the split partition is not unique, but that there are at most $|V(G)|$ many different split partitions.

A different characterization of split graphs is that they are the class of $\mathcal{H}$-free graphs [BLS99], where $\mathcal{H} = \{2K_2, C_4, C_5\}$. One very intriguing property of split graphs is that the editing number, or the splittance of a graph, is linear time computable [HS81]. That is, Split Editing is in P. It is interesting to note that this class is closed under complement; If you have a split graph G, then also the complement $\overline{G}$ is a split graph. The easiest way to see this is to consider a split partition (C, I) of G and observing that (I, C) will be a split partition in $\overline{G}$. A different way of verifying this is to observe that $\overline{2K_2} = C_4$ and that $\overline{C_5} = C_5$.

We say that a split graph $G = (V, E)$ with split partition (C, I) is *sparse* if $E = [C]^2$, and *dense* if $E = [V]^2 \setminus [I]^2$. That is, G is sparse if it is a disjoint union of a clique and an independent set, and dense if it is the complement, that is, a complete join of a clique and an independent set. The problem to editing to a sparse split graph has in the literature been called Clique Editing [DM14a, KSS14].

The *pseudosplit* graphs are mostly interesting as an artifact of the split graphs and the finite characterization; When considering the forbidden graphs of the split graphs, $\{2K_2, C_4, C_5\}$, a natural question is, what happens when we remove the peculiar occurrence of the five-cycle? It turns out that this yields a rather strange graph class. Pseudosplit graphs are the $\{2K_2, C_4\}$-free graphs.

A split graph is a pseudosplit graph. This is trivial by the fact that it is $\{2K_2, C_4\}$-free. However, the *imperfect pseudosplit graphs* [BHPT93, MP94], are the graphs whose vertex set can be partitioned into three sets (C, S, I) such that C is a clique, I is an independent set and $S \cong C_5$ and S is completely joined to C and there are no edges between S and I.

It turns out, by Maffray and Preissman [MP94], that also imperfect pseudosplit graphs are characterized by degree sequences. A graph $G = (V, E)$ with degree sequence d is imperfect pseudosplit if and only if

$$\sum_{i=1}^{p} d_i = p(p+4) + \sum_{i=p+6}^{n} d_i, \text{ and}$$

$d_{q+1} = d_{q+2} = \cdots = d_{q+5} = p + 2$, where $p = \max\{i \mid d_i \geq i + 4\}$.

Proposition 2.19 ([MP94])**.** *A graph $G = (V, E)$ is pseudosplit if and only if one of the following holds*

- *G is a split graph, or*
- *V can be partitioned into a pseudosplit partition (C, I, X) such that $G[C \cup I]$ is a split graph with C being a clique and I being an independent set, $G[X] \cong C_5$,*

and moreover, there is no edge between X and I and every edge is present between X and C.

Note that pseudosplit graphs are *self-complementary*. This can be observed by either observing that the forbidden set $\{2K_2, C_4\}$ is self-complementary, or by observing that for a given C, S, I-partitioning of G, the complement $\overline{C}, \overline{S}, \overline{I}$ will have $\overline{G[S]} \cong C_5$, $\overline{G[C]}$ an independent set, $\overline{G[I]}$ a clique, and all edges will be present between S and I and no edge present between S and C.

It is also worth noting that, just like split graphs [HS81], the pseudosplit graphs are characterized by a degree sequence [MP94]. Indeed, the pseudosplit graphs are *degree-sequence-forcing* [BKH08], that is, if a realization of a degree sequence $\mathbf{d}$ is a pseudosplit graph, then every realization of $\mathbf{d}$ is a pseudosplit graph. This also holds for split graphs and threshold graphs [MP95].

Cographs and trivially perfect graphs. The *cographs* are precisely the graphs that are P_4-free. The name comes from "complement reducible" [CLB81].

- K_1 is a cograph,
- $G_1 \uplus G_2$ is a cograph if both G_1 and G_2 are, and
- $G_1 \bowtie G_2$ is a cograph if both G_1 and G_2 are.

The *trivially perfect graphs* are the closures of rooted trees. This graph class is sometimes referred to as *quasi-threshold graphs* [JHJJC96, NG13], or as intersection graphs of nested intervals [BLS99]. Let T be a rooted tree, and for every vertex u and v that are on a path from the root to a leaf, add the edge uv. The resulting graph is called the closure of T. A graph G is a trivially perfect graph if and only if it is the closure of a tree. Trivially perfect graphs happen also to be intersection graph of *laminar* intervals (see Definition 2.10). The trivially perfect graphs can also be characterized as the $\{C_4, P_4\}$-free graphs, or by the following inductive definition [BLS99]:

- K_1 is a trivially perfect graph,
- $G_1 \uplus G_2$ is a trivially perfect graph is G_1 and G_2 are, and
- $G \bowtie K_1$ is a trivially perfect graph if G is.

Since the trivially perfect graphs (and next up, the threshold graphs) are the main graph classes with which we will be dealing in this thesis, we will spend some extra time diving into the structure of trivially perfect graphs. These observations and results will be crucial for the proof of Theorem 10, that there is a subexponential time algorithm solving Trivially Perfect Completion, in Chapter 10.

Threshold and chain graphs. The threshold graphs form a large subclass of the trivially perfect graphs and split graphs, in that they are the P_4-free split graphs. That is, they are $\{2K_2, C_4, P_4\}$-free. Originally, the name was chosen because of the following characterization of the graph class. A graph $G = (V, E)$ is threshold if and only if there exists an assignment of positive real numbers $r\colon V \to \mathbb{R}_+$ and a real number t, called the *threshold*, such that $uv \in E$ if and only if $r(u) + r(v) \geq t$.

Proposition 2.20 ([MP95])**.** *A graph G is a threshold graph if and only if G has a split partition (C, I) and the neighborhoods of the vertices of I are nested.*

Proposition 2.21 ([BLS99])**.** *A graph G is a threshold graph if and only if G does not have a C_4, P_4 nor a $2K_2$ as an induced subgraph. Thus, the threshold graphs are exactly the $\{2K_2, C_4, P_4\}$-free graphs.*

Chain graphs are the bipartite graphs whose neighborhoods of the vertices on one of the sides form an inclusion chain. It follows that the neighborhoods on the opposite side form an inclusion chain as well. If this is the case, we say that the neighborhoods are *nested*. The relation to threshold graphs is obvious, see Figure 2.4 for a comparison; A graph is a chain graph if and only if the graph by making one of the bipartitions into a clique is a threshold graph. The problem of completing edges to obtain a chain graph was introduced by Golumbic [Gol80] and later studied by Yannakakis [Yan81a], Feder, Mannila and Terzi [FMT09] and finally by Fomin and Villanger [FV13] who showed that Chain Completion when given a bipartite graph whose bipartition must be respected is solvable in subexponential time. We remove the latter requirement in Section 9.2.

A formal definition of the chain graph, that formalizes the nestedness property is the following, however, we also observe that the chain graphs are exactly the $\{2K_2, C_3, C_5\}$-free graphs.

Definition 2.22 (Chain graph)**.** A bipartite graph $G = (A, B, E)$ is a chain graph if there is an ordering of the vertices of A, $a_1, a_2, \ldots, a_{|A|}$ such that $N(a_1) \subseteq N(a_2) \subseteq \cdots \subseteq N(a_{|A|})$.

Biclusters and starforests. A *star graph* is a tree of diameter at most two (a graph isomorphic to $K_{1,\ell}$ for some ℓ). The degree-one vertices are called *leaves* and the vertex of higher degree, or the unique isolated vertex in the case of $S_1 = K_{1,0}$ the *center*. In the case of the edge, the graph $K_{1,1} = S_2 = P_2 = K_2$, we will fix one to be the center and the other the leaf. A *starforest* is a forest whose connected components are stars or, equivalently, a graph that does not contain $\{C_4, K_3, P_4\}$ as induced subgraphs. A *biclique* is a complete bipartite graph $K_{a,b}$ for some $a, b \in \mathbb{N}$, and a *bicluster graph* is a disjoint union of bicliques. That is, a graph is a *biclique* if it is a bipartite graph with bipartition (A, B) and for every pair of vertices $v_1 \in A$ and $v_2 \in B$, there is an edge v_1v_2. We write $K_{a,b}$ for the complete bipartite graph with a vertices in A and b vertices in B. A bicluster graph is a graph that is a disjoint union of bicliques. A t-partite clique graph is a graph whose vertex set

can be partitioned into at most t independent sets, all pairwise fully connected, and a t-partite cluster graph is a disjoint union of t-partite cliques.

2.2.3 Structure of trivially perfect graphs

Apart from the aforementioned characterization by forbidden induced subgraphs, an inherently local characterization, several other equivalent definitions of trivially perfect graphs are known. These definitions reveal more structural properties of this graph class which will be essential in our algorithm. We now establish a number of results on the global structure of trivially perfect graphs and minimal completions which will be useful.

The trivially perfect graphs have a rooted decomposition tree, which we call a *universal clique decomposition*, in which each node corresponds to a maximal set of vertices that all are universal for the graph induced by the vertices in the subtree rooted at this node. This decomposition is similar to that of a *treedepth decomposition*. We refer to Figure 2.2 for an example of the concepts that we introduce next. The following recursive definition is often used as an alternative definition of trivially perfect graphs.

Proposition 2.23 ([JHJJC96])**.** *The class of trivially perfect graphs can be defined recursively as follows:*

- K_1 *is a trivially perfect graph.*
- *Adding a universal vertex to a trivially perfect graph results in a trivially perfect graph.*
- *The disjoint union of two trivially perfect graphs is a trivially perfect graph.*

Let T be a rooted tree and t be a node of T. We denote by T_t the maximal subtree of T rooted in t. We can now use the universal clique $\mathrm{uni}(G)$ of a trivially perfect graph $G = (V, E)$ to make a decomposition structure.

Definition 2.24 (Universal clique decomposition)**.** A *universal clique decomposition*, or a *UCD*, of a connected trivially perfect graph $G = (V, E)$ is a pair $(T = (V_T, E_T), \mathcal{B} = \{B_t\}_{t \in V_T})$, where T is a rooted tree and $\mathcal{B}$ is a partition of the vertex set V into disjoint non-empty subsets, such that

- if $vw \in E(G)$ and $v \in B_t$ and $w \in B_s$, then s and t are on a path from a leaf to the root, with possibly $s = t$, and
- for every node $t \in V_T$, the set of vertices B_t is the maximal universal clique in the subgraph $G[\bigcup_{s \in V(T_t)} B_s]$.

We call the vertices of T *nodes* and the sets in $\mathcal{B}$ *bags* of the universal clique decomposition $(T, \mathcal{B})$. By slightly abusing the notation, we often do not distinguish between nodes and bags. Note that by the definition, in a universal clique decomposition every non-leaf node has at least two children, since otherwise the universal clique contained in the corresponding bag would not be maximal.

Lemma 2.25. *A connected graph G admits a universal clique decomposition if and only if it is trivially perfect. Moreover, such a decomposition is unique up to isomorphisms.*

Proof. From right to left, we proceed by induction on the number of vertices using Proposition 2.23. The base case is when we have one vertex, K_1 which is a trivially perfect graph and also admits a unique universal clique decomposition. The induction step is when we add a vertex v, and by the definition of trivially perfect graphs, v is a universal vertex. Either we add a universal vertex to a connected trivially perfect graph, in which case we simply add the vertex to the root bag, or we add a universal vertex to the disjoint union of two or more trivially perfect graphs. In this case, we create a new tree, with r_v being the root connected to the root of each of the trees for the disjoint union. Since v is the only universal vertex in the graph, the constructed structure is a universal clique decomposition. Observe that the constructed decompositions are unique (up to isomorphisms).

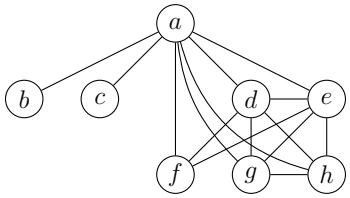
From left to right, we proceed by induction on the height of the universal clique decomposition. Suppose $(T, \mathcal{B})$ is a universal clique decomposition of a graph G. Consider the case when T has height 1, i.e., we have only one single tree node (and one bag). Then this bag, by Proposition 2.23, is a clique (every vertex in the bag is universal), and since a complete graph is trivially perfect, the base case holds. Consider now the case when T has height at least 2. Let r be the root of T, and let $x_1, x_2, \ldots, x_p$ be children of r in T. Observe that the tree T_{x_i} is a universal clique decomposition for the graph $G[\bigcup_{t \in V(T_{x_i})} B_t]$ for each $i = 1, 2, \ldots, p$. Hence, by the induction hypothesis we have that $G[\bigcup_{t \in V(T_{x_i})} B_t]$ is trivially perfect. To see that G is trivially perfect as well, observe that G can be obtained by taking the disjoint union of graphs $G[\bigcup_{t \in V(T_{x_i})} B_t]$ for $i = 1, 2, \ldots, p$, and adding $|B_r|$ universal vertices. □

For the purposes of the dynamic programming procedure, we define the following notion.

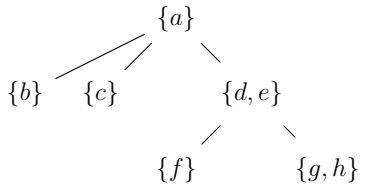
Definition 2.26 (Block)**.** Let $(T = (V_T, E_T), \mathcal{B} = \{B_t\}_{t \in V_T})$ be the universal clique decomposition of a connected trivially perfect graph $G = (V, E)$. For each node $t \in V_T$, we associate a *block* $L_t = (B_t, D_t)$, where

- B_t is the subset of V contained in the bag corresponding to t, and
- D_t is the set of vertices of V contained in the bags corresponding to the nodes of the subtree T_t.
- The *tail* of a block L_t is the set of vertices Q_t contained in the bags corresponding to the nodes of the path from t to r in T, where r is the root of T, including B_t and B_r.

When t is a leaf of T, we have that $B_t = D_t$ and we call the block $L_t = (B_t, D_t)$ a *leaf block*. If t is the root, we have that $D_t = V(G)$ and we call L_t the *root block*. Otherwise, we call L_t an *internal block*. Observe that for every block $L_t = (B_t, D_t)$



(a) A trivially perfect graph G.



(b) The universal clique decomposition of G with the bags as labels.

Label (bag)	Block	Tail
$\{a\}$	$(\{a\}, V)$	$\{a\}$
$\{b\}$	$(\{b\}, \{b\})$	$\{a, b\}$
$\{c\}$	$(\{c\}, \{c\})$	$\{a, c\}$
$\{d, e\}$	$(\{d, e\}, \{d, e, f, g, h\})$	$\{a, d, e\}$
$\{f\}$	$(\{f\}, \{f\})$	$\{a, d, e, f\}$
$\{g, h\}$	$(\{g, h\}, \{g, h\})$	$\{a, d, e, g, h\}$

(c) Table of the bags with corresponding blocks and tails.

Figure 2.2: In the first figure, we have a trivially perfect graph, and in the second, a *universal clique decomposition* of the graph with the bags as labels. Finally we have a table of the bags and the corresponding blocks and tails. Notice that for a block (B, D) and tail Q, $B \subseteq D$ and $B \subseteq Q$. Furthermore, in any leaf block it holds that $B = D$, and in the root block it holds that $D = V$.

with tail Q_t we have that $B_t \subseteq Q_t$, $B_t \subseteq D_t$, and $D_t \cap Q_t = B_t$, see Figure 2.2. Note also that Q_t is a clique and the vertices of Q_t are universal to $D_t \setminus B_t$. The following lemma summarizes the properties of universal clique decompositions, maximal cliques, and blocks used in our proof.

Lemma 2.27. *Let $(T, \mathcal{B})$ be the universal clique decomposition of a connected trivially perfect graph G and let $L = (B, D)$ be a block with Q as its tail.*

(i) The following are equivalent:

(1) L is a leaf block,

(2) $D = B$,

(3) Q is a maximal clique of G, and

(4) $Q = N_G[v]$ for every $v \in B$.

(ii) If L is a non-leaf block, then for every two vertices u, v from different connected components of $G[D \setminus B]$, we have that $Q = N_G(u) \cap N_G(v)$.

(iii) For every maximal clique Q' of G, there exists a leaf block L' in $(T, \mathcal{B})$ with tail Q'.

Proof. (i) We first prove the chain (1) $\to$ (2) $\to$ (3) $\to$ (1). If L is a leaf block and D is the set of vertices in all the bags of the subtree rooted at L, then $B = D$. If $D = B$ we have that $N_G[v] = Q$ for any $v \in B$ (so also (2) $\to$ (4)). Hence Q is maximal. If L is not a leaf block, then there is a vertex v which belongs to a child node of L, and for any such vertex v the set $Q \cup \{v\}$ is a clique, and so Q is not a maximal clique. Hence if Q is a maximal clique, then L is a leaf block.

Finally, we show that (4) $\to$ (2): Since Q is a clique and $v \in Q$, we have that $Q \subseteq N_G[v]$. On the other hand, since $v \in B$ and L is a leaf block, we have that $Q \supseteq N_G[v]$ by the definition of universal clique decomposition.

(ii) Suppose $L = (B, D)$ is a non-leaf block and D_1 and D_2 are two connected components of $G' = G[D \setminus B]$. Let $v \in D_1$ and $u \in D_2$ and observe that since they are in different connected components of $G[D \setminus B]$, $N_{G'}(v) \cap N_{G'}(u) = \emptyset$. By the universality of Q, the result follows: $Q = N_G(v) \cap N_G(u)$.

(iii) Let $(T, \mathcal{B})$ be a UCD and consider the following construction of a tail of a block L_t. Obviously $\text{uni}(G) \subseteq Q'$. Let $L_1 = (\text{uni}(G), V(G))$. Consider the disconnected trivially perfect graph $G - \text{uni}(G)$. This has exactly one component containing vertices from Q'. Let G_2 be the connected component of $G - \text{uni}(G)$ containing a vertex from Q'. Again $\text{uni}(G_2) \subseteq B'$, so let $L_2 = (\text{uni}(G_2), V(G_2))$. Continue until L_t is a leaf bag. Since $\text{uni}(G_t) = V(G_t)$ and since $\text{uni}(G_t) \subseteq Q'$ (again by maximality of Q'), we have that $Q' = \bigcup_{1 \leq 1 \leq t} \text{uni}(G_i)$, which is exactly the tail of L_t. Since $(T, \mathcal{B})$ is unique, L_t is a leaf bag of $(T, \mathcal{B})$ which proves the claim. □

2.2.4 Structure of threshold and chain graphs

In this section, we highlight some of the similarities between threshold graphs and chain graphs. As we did for the trivially perfect graphs above, we will give a decomposition theorem for threshold graphs.

Definition 2.28 (Threshold partition)**.** We say that a partition of the vertex set into $(\mathcal{C}, \mathcal{I})$, where $\mathcal{C} = \langle C_1, \ldots, C_t \rangle$, and $\mathcal{I} = \langle I_1, \ldots, I_t \rangle$), forms a *threshold partition* of G if the following holds (see Figure 2.3 for an illustration):

- (C, I) is a split partition of G, where $C = \bigcup_{i \leq t} C_i$ and $I = \bigcup_{i \leq t} I_i$,
- C_i and I_i are twin classes in G for every i
- $N[C_j] \subset N[C_i]$ and $N(I_i) \subset N(I_j)$ for every $i < j$.
- Finally, we demand that for every $i \leq t$, $(C_i, I_{\geq i})$ form a complete split partition of the graph induced by $C_i \cup I_{\geq i}$.

We furthermore define, for every vertex v in G, $\text{lev}(v)$ as the number i such that $v \in C_i \cup I_i$ and we denote each level $L_i = C_i \cup I_i$.

In a threshold decomposition we will refer to C_i for every i as a *clique fragment* and I_i as a *independent fragment*. Furthermore, we will refer to a vertex in $\cup\mathcal{C}$ as a *clique vertex* and a vertex in $\cup\mathcal{I}$ as an *independent vertex*.

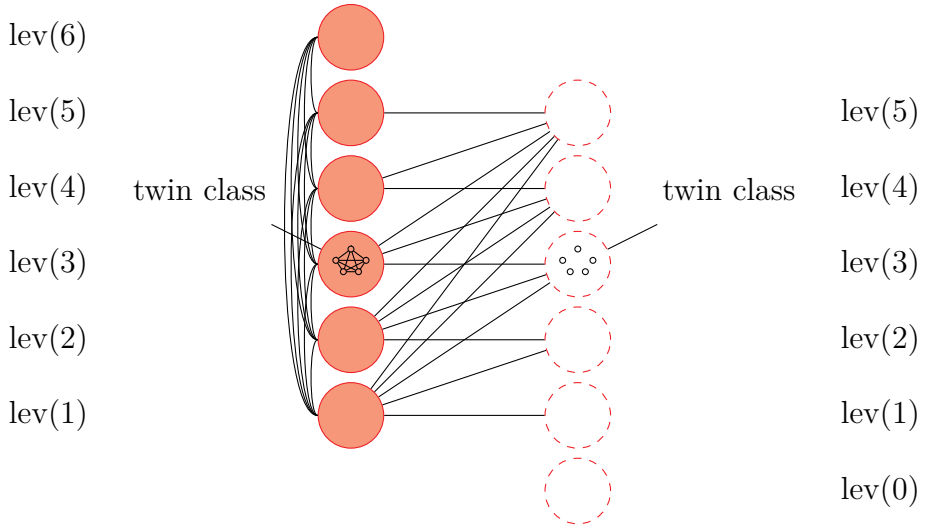
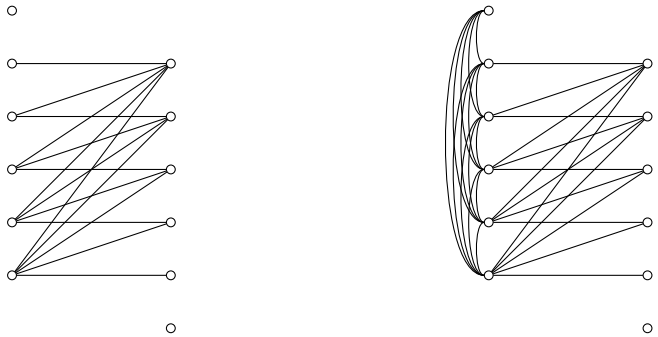


Figure 2.3: A threshold partition—the left hand side is the clique and the right hand side is an independent set, each bag contains a twin class. All bags are non-empty, otherwise two twin classes on the opposite side would collapse into one, except possibly the two extremal bags.

Proposition 2.29 (Threshold decomposition)**.** *A graph G is a threshold graph if and only if G admits a threshold partition.*

Proof. Suppose that G is a threshold graph and therefore admits a nested ordering of the neighborhoods of vertices of each side [HIS81]. We show that partitioning the graph into partitions depending only on their degree yields the levels of a threshold partition. The clique side is naturally defined as the maximal set of highest degree vertices that form a clique. Suppose now for contradiction that this did not constitute a threshold partition. By definitions, every level consists of twin classes, and also, for two twin classes I_i and I_j, since their neighborhoods are nested in the threshold graph, their neighborhoods are nested in the threshold partition as well. So what is left to verify is that $(C_i, I_{\geq i})$ is a complete split partition of $G[C_i \cup I_{\geq i}]$. But that follows directly from the assumption that G admitted a nested ordering and C_i is a true twin class.

For the reverse direction, suppose G admits a threshold partition $(\mathcal{C}, \mathcal{I})$. Consider any four connected vertices a, b, c, d. We will show that they can not form any of the induced obstructions. For the $2K_2$ and C_4, it is easy to see that at most two of the vertices can be in the clique part of the decomposition—and they must be adjacent since it is a clique—and hence there must be an edge in the independent set part of the decomposition, which contradicts the assumption that $\mathcal{C}, \mathcal{I}$ was a threshold partition. So suppose now that a, b, c, d forms a P_4. Again with the same reasoning as above, the middle edge b, c must be contained in the clique part, hence a and d must be in the independent set part. But since the neighborhoods of a and d should be nested, they cannot have a private neighbor each, hence either ac or bd must be an edge, which contradicts the assumption



(a) A chain graph (b) A threshold graph

Figure 2.4: The relationship between a chain and a threshold graph.

that a, b, c, d induced a P_4. This concludes the proof. □

Lemma 2.30. *For any instance (G, k) of editing or completing to either threshold or chain graphs, it holds that there exists an optimal solution F such that for every pair of vertices $u, v \in V(G)$, if $N_G(u) \subseteq N_G[v]$ then $N_{G \triangle F}(u) \subseteq N_{G \triangle F}[v]$.*

Proof. Let us define, for any editing set F and two vertices u and v, the set

$$F_{v \leftrightarrow u} = \{e \mid e' \in F \text{ and } e \text{ is } e' \text{ with } u \text{ and } v \text{ switched}\}.$$

Suppose F is an optimal solution for which the above statement does not hold. Then $N_G(u) \subseteq N_G[v]$ and $N_{G \triangle F}(v) \subseteq N_{G \triangle F}[u]$ (see Proposition 2.20). But then it is easy to see that we can flip edges in an ordering such that at some point, say after flipping F^0, u and v are twins in this intermediate graph $G \triangle F^0$. Let $F^1 = F \setminus F^0$. It is clear that for $G' = G \triangle (F^0 \cup F^1_{v \leftrightarrow u})$, $N_{G'}(u) \subseteq N_{G'}[v]$. Since $|F| \geq |F^0 \cup F^1_{v \leftrightarrow u}|$, the claim holds. □

From the following proposition, it follows that chain graphs are characterized by a finite set of forbidden induced subgraphs and hence are subject to Cai's theorem [Cai96].

Proposition 2.31 ([BLS99])**.** *Let G be a graph. The following are equivalent:*

- *G is a chain graph.*
- *G is bipartite and $2K_2$-free.*
- *G is $\{2K_2, C_3, C_5\}$-free.*
- *G can be constructed from a threshold graph by removing all the edges in the clique partition.*

Since they have the same structure as threshold graphs, it is natural to talk about a *chain decomposition*, $(\mathcal{A}, \mathcal{B})$ of a bipartite graph G with bipartition (A, B). We say that $(\mathcal{A}, \mathcal{B})$ is a chain decomposition for a chain graph G if and only if $(\mathcal{A}, \mathcal{B})$ is a threshold decomposition for the corresponding threshold graph G' where A is made into a clique.

2.2.5 Small graphs

In this section we simply list some small graphs that appear throughout this thesis. These graphs are small but fundamental building blocks, or fundamental obstructions, in graphs and graph classes we are concerned with in this thesis. Of special importance are the paths and cycles on four vertices, as well as $2K_2$, the complement of the four-cycle.

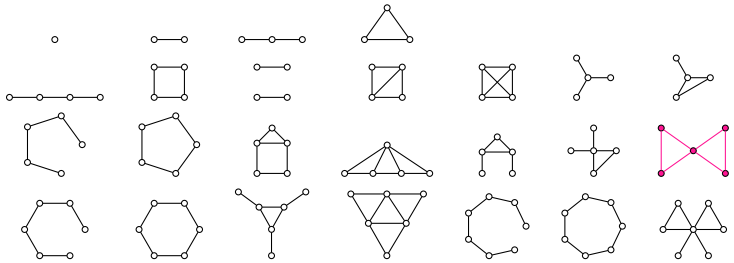


Table 2.1: Table of small graphs.

In the table above, the following graphs are listed:

Row 1: K_1, $K_2 = P_2$, P_3, $K_3 = C_3$.
Row 2: P_4, C_4, $\overline{C_4} = 2K_2$, diamond, K_4, $K_{1,3} = S_4 =$ claw, paw.
Row 3: P_5, C_5, house $= \overline{C_5}$, gem, bull, cricket, butterfly.
Row 4: P_6, C_6, net, sun, P_7, C_7, H_{KW}.

2.3 Parameterized complexity

The primary tool for studying graph modification problems has in the work leading to this thesis been *parameterized complexity*. Parameterized complexity is a field introduced by Downey and Fellows [DF99]. For an introduction to this field, we refer the reader to the original work and to Flum and Grohe [FG06].

In parameterized complexity, we introduce a new measure, a *parameter* to the problem instance, whose purpose is to extract information on where the complexity of the problem truly lies. A problem is said to be fixed parameter tractable with respect to a certain parameter k if there exists a function f such that any instance x of the problem, with parameter k can be solved in time $f(k) \cdot \text{poly}(|x|)$ where poly is any polynomial and $|x|$ denotes the total length of the input. We define the complexity class FPT to be the class of all problems solvable in fixed-parameter tractable time.

One of the main advantages of analyzing problems from a parameterized, or *multivariate* perspective is that it helps us in creating fast exponential algorithms. Before the invention of parameterized algorithms, the only way of measuring the complexity was as a function of the size of the input. When a problem is NP-hard, we often do not expect any algorithms to solve our instance much faster than c^n for some constant $c > 1$. In the light of parameterized complexity, we introduce a secondary measure—the parameter—which we use to measure the complexity

of the instance. In fixed-parameterized time, we allow a polynomial time in the length of the input, and exponential time in the parameter.

A second benefit of parameterized complexity is that as we observed, there are problems that are unlikely to be fixed-parameter tractable, and this gives rise to a whole new hierarchy of computational complexity. There are three fundamental basic graph problems, Clique and Independent Set, which are both W[1]-complete, and Dominating Set which is W[2]-complete. Although the so called W-hardness is outside the scope of this thesis[3], since all the problems considered in this thesis are in FPT by a result of Cai [Cai96], it is important to know that there are problems that are unlikely to be in FPT. By "unlikely", here we mean that if $\mathsf{FPT} = \mathsf{W[1]}$, then $\mathsf{NP} \subseteq \mathsf{DTIME}(2^{o(n)})$, and in particular, ETH would fail.

A *parameterized problem* is a subset $Q \subseteq \Sigma^* \times \mathbb{N}$ for some finite alphabet Σ, where the second part of the input is called the *parameter*. A parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ is said to be *fixed-parameter tractable* if for each pair $(x, k) \in \Sigma^* \times \mathbb{N}$ it can be decided in time $f(k) \cdot \text{poly}(|x|)$ whether $(x, k) \in Q$, for some function f that only depends on k. Here $|x|$ denotes the length of the input x.

For all our purposes in this thesis, the full language Σ^* will be that of undirected simple graphs, denoted $\mathcal{G}$, and thus in most our problems, the input instances will be of the form (G, k) where G is any finite simple graph. Hence, fixed-parameter tractable algorithms will in this case have running times on the form $f(k) \cdot \text{poly}(n)$.

Definition 2.32 (SUBEPT)**.** The complexity class SUBEPT is defined as all the problems Q that can be solved in time $2^{k/s(k)} \text{poly}(n, k) = 2^{o(k)} \text{poly}(n, k)$, where s is a monotonically increasing unbounded function.

Proposition 2.33 ([FG06, LMS11])**.** *A parameterized problem Q is in* SUBEPT *if and only if there is an algorithm that for every $\epsilon > 0$ solves the instances (G, k) of Q in time $2^{\epsilon k} \text{poly}(n, k)$.*

2.3.1 Polynomial kernels

One of the main tools that parameterized complexity brought the field of algorithms and especially the field of algorithmic graph theory, is the concept of a kernel. Intuitively, a kernelization algorithm is an algorithm that in polynomial time preprocesses—or compresses—a given instance and outputs an instance whose size is bounded polynomially in the parameter, or as is most often in our case, in the solution size. The goal of kernelization is (at least) two-fold. First, it is useful as a preprocessing stage, as we now may assume with polynomial overhead that our instances are bounded polynomially in k, which allows us to do exponential work on the kernelized instance and still be within fixed-parameter tractable time. This can also be useful in intermediate steps in branching algorithms [AK15]. Second, the kernel-size of different problems is a form of complexity measure that combined

[3]Just barely outside, since Lokshtanov proved that Wheel-free deletion—a specific $\mathcal{H}$-free Deletion problem for an infinite set $\mathcal{H}$—is W[2]-complete [Lok08].

with provable lower bounds, e.g., on the assumption that $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$, gives rise to an optimality programme of compression.

More formally, a kernelization algorithm (often referred to as the kernel), is an algorithm for a problem that on an input (G, k) outputs an *equivalent instance* (G', k') with $\max\{|G'|, k'\} \leq f(k)$ for some function f. Furthermore, the algorithm needs to run in polynomial time.

Definition 2.34. We say that a parameterized problem Q has a *kernel* if there is an algorithm that transforms each instance (x, k) in time $\text{poly}(|x| + k)$ into an instance (x', k'), such that $(x, k) \in Q$ if and only if $(x', k') \in Q$ and $|x'| + k' \leq g(k)$ for some function g. If g is a polynomial, then we say that the problem admits a *polynomial* kernel.

If in addition to g being a polynomial, we are guaranteed that $k' \leq k$, we say that the kernel is a *proper polynomial kernel*. The following result shows that the power of kernels is, so to speak, equivalent to the power of fixed parameter tractability:

Proposition 2.35 ([DFS99, Lemma 4.8])**.** *A problem is fixed parameter tractable if and only if it admits a kernel and is decidable.*

Kernelization lower bounds. It turns out that not every problem, up to certain complexity assumptions admits a polynomial kernel. The following might serve as an intuition for that case. Consider the problem k-Path, which is the following:

k-Path parameterized by k

Input:	A graph G and an integer k
Question:	Does G have a path on k vertices?

This problem is NP-complete and solvable in fixed-parameter time (thus in FPT), however it is unlikely to have a polynomial kernel. Suppose it had a polynomial kernel of size k^c for some c. Suppose furthermore that you had $k^{c^{100}}$ different instances, $G_1, G_2, \ldots, G_{k^{c^{100}}}$, all with the same parameter k. Now, let us construct $\hat{G} = \biguplus_{i \leq k^{c^{100}}} G_i$, the disjoint union of all the above graph instances, and furthermore observe that $\hat{G}$ has a path on k vertices if and only if at least for one $i \leq k^{c^{100}}$, G_i has a path on k vertices.

Let (G', k') be the output instance of the alleged kernelization algorithm on input $(\hat{G}, k)$, and observe that the size of G' and k' are both bounded by k^c. Even in bits, this is much smaller than just the number of input instances, so it cannot even store one bit per input instance. That means that the polynomial time algorithm must have been able to rule out many of the instances, each of which could have been an important yes-instance.

To paraphrase Ryan Williams [Wil11], "at this point we have reached a degree of handwaving so exuberant, one may fear we are about to fly away. Surprisingly, this handwaving has a completely formal theory behind it," and that is the theory

we will now briefly discuss. Namely, that this can be formalized in a way that shows that if k-PATH had a polynomial kernel, the polynomial hierarchy would collapse to the second level, or more specifically, $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

The lower bounds result in Section 8.2 will be based on the OR-cross-composition framework by Bodlaender, Jansen, and Kratsch [BJK14], which builds on previous work by Bodlaender, Downey, Fellows, and Hermelin [BDFH09], Fortnow and Santhanam [FS11], and Dell and van Melkebeek [DM14b]. This framework provides nice tools for proving polynomial kernel impossibility results.

The following framework for lower bounds, the OR-cross-composition framework is taken verbatim from the article by Bodlaender, Jansen, and Kratsch:

Definition 2.36 (OR-cross-composition, [BJK14])**.** Let $L \subseteq \Sigma^*$ be a language, $\mathcal{R}$ a polynomial equivalence relation on Σ^*, and let $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. An OR-*cross-composition* of L into $\mathcal{Q}$ (with respect to $\mathcal{R}$) is an algorithm that, given instances $x_1, x_2, \cdots, x_t \in \Sigma^*$ of L belonging to the same equivalence class of $\mathcal{R}$, takes time polynomial in $\sum_{i=1}^{t} |x_i|$ and outputs an instance $(y, k) \in \Sigma^* \times \mathbb{N}$ such that

"PB": The parameter value k is polynomially bounded in $\max_{i=1}^{t} |x_i| + \log t$.

"OR": The instance (y, k) is a yes-instance for $\mathcal{Q}$ if and only if at least on instance x_i is a yes-instance for L.

Proposition 2.37 ([BJK14])**.** *If L is an* NP*-complete language and has an* OR*-cross-composition into a parameterized language $\mathcal{Q}$, then $\mathcal{Q}$ does not have a polynomial compression, provided* $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$.

2.3.2 Lower bounds using ETH

In this section we give the main tools for the lower bounds used in Part IV. The main ingredients here are the exponential time hypothesis and the sparsification lemma, both by Impagliazzo, Paturi, and Zane [IPZ01]. For an introduction to lower bounds using ETH, see the excellent survey of Lokshtanov, Marx, and Saurabh [LMS11].

As far as we know, problems that are NP-complete *might* be solvable in polynomial time. It is considered highly unlikely by most computer scientists and mathematicians around the globe, but as long as we do not have a proof that P is not equal to NP, we cannot rule out such algorithms. What's worse is that we are very—*very*— far away from being able to show this; We do not even know if P is a proper subset of PSPACE.

However, although the theory behind NP-hardness gives us strong evidence that certain problems are not solvable in polynomial time, we cannot use this theory to rule out algorithms of the form $n^{O(\log n)}$ (so-called quasi-polynomial running time), or $2^{o(n)}\,\mathrm{poly}(n)$. However, attempts on finding algorithms computing, e.g., the chromatic number of a graph in subexponential time $2^{o(n)}$ has not given positive results.

Thus, for some problems, we would like to show that we need truly exponential time, for example, for 3Sat (defined below), it seems plausible that it should be impossible to solve φ in $2^{o(|\mathcal{V}(\varphi)|)}\operatorname{poly}(|\varphi|)$ time, where $\mathcal{V}_\varphi$ is the set of variables of φ. However, as this would imply that $\mathsf{P} \neq \mathsf{NP}$, we cannot hope for such results yet. This led Impagliazzo, Paturi and Zane [IPZ01] to define a new complexity theoretic hypothesis—*the exponential time hypothesis*—which is slightly stronger than $\mathsf{P} \neq \mathsf{NP}$. Whether or not one believes that the hypothesis is true or not is actually not that important, for as Lokshtanov et al. [LMS11] notes,

> the very least, these results show that breaking the lower bounds require fundamental advances in satisfiability algorithms, and therefore problem-specific ideas related to the particular problem are probably not of any help.

Before we state the hypothesis, we formally define some notions on satisfiability and reductions needed for the hypothesis. The results below hold for all $r \geq 3$, but for simplicity, we will only refer to 3-CNF-SAT, rather than the more general r-CNF-SAT.

Definition 2.38 (3-CNF-SAT)**.** A formula φ over variables $\mathcal{V}_\varphi$ and clauses $\mathcal{C}_\varphi$ is said to be a 3-CNF-SAT formula if every clause contains at most three variables. An assignment for a SAT formula is a function $\alpha\colon \mathcal{V}_\varphi \to \{\texttt{true}, \texttt{false}\}$. We say that α *satisfies* φ if for every clause $c \in \mathcal{C}_\varphi$, there is a variable which α-satisfies c. A SAT formula φ is *satisfiable* if it admits a satisfying assignment.

3Sat

Input:	A 3-CNF-SAT formula φ
Question:	Is φ satisfiable?

Exponential time hypothesis (ETH). *There exists a positive real number s such that* 3Sat *with n variables cannot be solved in time 2^{sn}.*

Impagliazzo et al. [IPZ01] defined the notion of a *subexponential reduction family* (SERF) to construct lower bounds using ETH. However, for this thesis we only need the less general and slightly simpler class of reductions which are called FPT-reductions with linear parameter blow-up.

Definition 2.39 (FPT-reduction, [LMS11])**.** An FPT-reduction from Π_1 to Π_2 is a mapping $R : \mathcal{G} \times \mathbb{N} \to \mathcal{G} \times \mathbb{N}$ with a corresponding computable function g, such that

- for all (G, k), $(G, k) \in \Pi_1$ if and only if $R((G, k)) \in \Pi_2$,
- R is computable by an FPT-algorithm parameterized by k, and
- for any instance (G, k), $k' \leq g(k)$, where k' is the parameter output by R.

It g is linear, we say that the reduction has a linear parameter blow-up.

It can be observed that if there is an FPT-reduction from Π_1 to Π_2 with a linear parameter blow-up, then if $\Pi_2 \in \mathsf{SUBEPT}$ then $\Pi_1 \in \mathsf{SUBEPT}$.

Since, in a 3-CNF-SAT formula φ, every variable occurs in at least one clause, it follows that $|\mathcal{V}| \leq 3|\mathcal{C}|$. An immediate consequence is that 3SAT with parameter (complexity measure) $|\mathcal{C}|$ (the number of clauses) is SERF-reducible to 3SAT with parameter $|\mathcal{V}|$. The reverse direction, however, is less clear. However, Impagliazzo et al. gave a SERF-reduction for the opposite direction as well, thereby showing that not even "sparse" instances are solvable in subexponential time, under ETH. This result is called the sparsification lemma. An instance of 3SAT is called *sparse* if it has $O(|\mathcal{V}|)$ many clauses, i.e., $|\mathcal{C}| = O(|\mathcal{V}|)$.

Proposition 2.40 (Sparsification lemma [IPZ01])**.** *For every $\epsilon > 0$ and positive integer r, there is a constant $C = O((\frac{r}{\epsilon})^{3r})$ so that any 3-CNF-SAT formula F with n variables, can be expressed as $F = \bigvee_{i=1}^{t} Y_i$, where $t \leq 2^{\epsilon n}$ and each Y_i is a 3-CNF-SAT formula with every variables appearing in at most C clauses. Moreover, this disjunction can be computed by an algorithm running in time $2^{\epsilon n}\,\mathrm{poly}(n)$.*

Impagliazzo et al. give a SERF-reduction from 3SAT with parameter $|\mathcal{V}|$ to 3SAT with parameter $|\mathcal{C}|$. Thus the following proposition is a direct consequence of the sparsification lemma.

Proposition 2.41 ([IPZ01])**.** *Assuming ETH, there is a positive real s such that* 3SAT *with parameter $|\mathcal{C}|$ cannot be solved in time $O(2^{sm})$. That is, there is no $2^{o(m)}$ algorithm for* 3SAT *with parameter $|\mathcal{C}|$.*

The following lemma is the conclusion from the above exposition and will be our main tool for showing lower bounds for graph modification problems.

Lemma 2.42. *Assuming ETH, if there is an* FPT*-reduction from* 3SAT *to a parameterized graph problem Π with linear parameter blow-up, and the number of vertices, n and edges, m is linear in the size of the formula, then that problem cannot be solved in $2^{o(n+m)}\,\mathrm{poly}(n)$ time, nor $2^{o(k)}\,\mathrm{poly}(n)$ parameterized time.*

Part II

Kernels

This part is devoted to the polynomial *kernelizability* of edge modification problems. There are many reasons for studying polynomial kernelizability for its own sake, however, as we will see in Part III, one of the main tools for obtaining subexponential parameterized time algorithms is precisely to be able to start out with a kernelized instance.

Three of the problems concerned in this part, Threshold Editing, Chain Editing, and Trivially Perfect Editing were previously not known to be NP-hard, hence the question of polynomial kernelizability seems strange; maybe we could just as well have solved them completely in polynomial time. However, in Part IV, we show that they are indeed NP-complete, and hence the problem of kernelizability is a natural next step. The NP-completeness of Trivially Perfect Editing was shown independently by Nastos and Gao [NG13]. Their proof suffers an (at least) cubic blow-up and can therefore not be used to rule out a subexponential time algorithm. Our reduction in Chapter 14 will be linear, both in size and parameter, and shows therefore that we should not expect algorithms of running time $2^{o(k)} \cdot \text{poly}(n)$.

The main contribution of this part is a new technique for obtaining polynomial kernelization results for edge modification problems towards $\mathcal{H}$-free graphs. We describe this case, and provide a small example as a use case, in the first chapter of this part; Chapter 3 gives a very simple quadratic vertex kernel for Cluster Deletion using the modulator technique.

In Chapters 4 and 5 we obtain quadratic sized vertex cover for all the edge modification problems towards threshold graphs as well as the related class chain graphs. In Chapter 6 we show how to use the modulator to obtain a kernel with $O(k^7)$ vertices for all edge modification problems towards trivially perfect graphs. This is by far the most technical chapter of this part. In Chapter 7, we show that there is a simple kernel with $O(kp)$ vertices for p-Bicluster Editing, and a more general variant of the problem. This will be useful for our algorithmic results on that problem in the next part.

We conclude the part on kernels by adding a result concerning bounded degree input graphs in Chapter 8. This shows that problems like editing or deleting towards C_4-free graphs, which in general do not admit polynomial kernels [Cai12], do admit polynomial kernels when the input graphs have bounded degree. We will also see that we cannot expect to be able to obtain the same result for the completion variant; that is, there is a finite set $\mathcal{H}$ for which $\mathcal{H}$-free Completion does not admit a polynomial kernel, even on bounded degree input graphs, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. Lifting the former positive results to degenerate graphs is also not possible, as shown by Kratsch and Wahlström [KW13]; Although not explicitly stated in their paper, a careful analysis of their proof reveals that edge deletion to H_{KW}-free graphs (See Table 2.1) does not admit a polynomial kernel, even if the input graphs are 6-degenerate, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Chapter 3

Technique: Modulator driven kernels — a showcase

In this chapter we present the main technique of this part by giving a very simple quadratic kernel for Cluster Deletion. Let (G, k) be an instance to Cluster Deletion, and recall that Cluster Deletion is the problem of deleting edges to obtain a P_3-free graph. The technique consists mainly of a twin reduction rule, and sometimes other irrelevant vertex deletion rules. For this case, we use the following two rules:

Rule 3.1. *If a connected component C is a cluster, continue with $(G - C, k)$.*

Rule 3.2. *If $T = \mathrm{tc}(v)$ is a twin class of size at least $2k + 4$ with $v \in T$, continue with the instance $(G - v, k)$.*

Lemma 3.1. *Rules 3.1 and 3.2 are sound and can be exhaustively applied in polynomial time.*

Proof. The first rule is straight forward. For the second rule, let $T = \mathrm{tc}(v)$ be a twin class of v. First we observe that in polynomial time, we can either find such a set, or conclude that no such set exists. Since cluster graphs are hereditary, if (G, k) is a yes-instance, then $(G - v, k)$ is a yes-instance for any vertex v. So suppose that $(G - v, k)$ is a yes-instance for some v that Rule 3.2 applied to, and suppose for the sake of a contradiction that (G, k) is a no-instance. We assume for simplicity that Rule 3.1 did not apply to v, hence v is not an isolated vertex.

Let F be a solution to $(G - v, k)$. Now, since T has at least $2k + 4$ vertices, and F can only be incident to $2k$ vertices, there are three vertices in $G - v$ not incident with any edge in F; We call these vertices v_1, v_2, and v_3. Since v is not an isolated vertex, neither are any of the three above. But this means that $\{v_1, v_2, v_3\}$ is a clique, and so is $\{v, v_1, v_2, v_3\}$. Since these vertices are twins, the connected component of v is a clique, hence $G - F$ is a cluster graph, which contradicts the assumption that $(G - v, k)$ was a no-instance. □

The main tool of analysis is the following. We construct a set X as follows: If there is an induced P_3 on vertex set W_1 with at most one vertex in X, we add W_1

to X. If there is an induced P_3 on vertex set W_2 with exactly two endpoints in X and the middle vertex not in X, we add the middle vertex to X. That is, any obstruction without any edges in X will be put into X. We continue until no such obstructions exists.

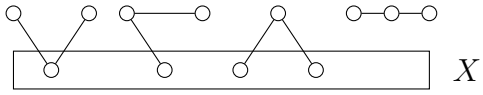


Figure 3.1: Forbidden intersections for obstructions. None of the four above configurations occurs after the algorithm above has been exhaustively applied.

Lemma 3.2. *If (G, k) is a yes-instance, $|X| \leq 3k$.*

Proof. Suppose that (G, k) is a yes-instance. We need only observe that every time we find an obstruction and add it to X, this is a witness that we need at least one more edge deletion to obtain a cluster graph. Hence in any yes-instance, this process must terminate after at most k steps. Since we never add more than three vertices at a time, $|X| \leq 3k$. □

Rule 3.3. *If $|X| > 3k$ continue with $(P_3, 0)$ (that is, we say no).*

We now claim that after the rules are applied, in any yes-instance, $|V(G)| = O(k^2)$. Since $G - X$ is a cluster graph and since every twin class has size at most $2k+3$, we have the following situation. Suppose a vertex $v \notin X$ has two neighbors $x_1, x_2 \in X$. Then $x_1x_2 \in E$ since otherwise $\{x_1, v, x_2\}$ would be an obstruction with only the endpoints in X. And if $v_1, v_2 \notin X$, with a common neighbor $x \in X$, then $v_1v_2 \in E$.

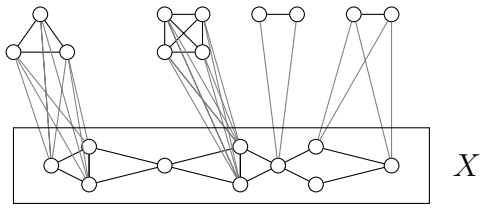


Figure 3.2: The kernelized instance.

This means that the neighborhoods of $G - X$ are partitioned into classes such that every component in $G - X$ sees exactly the same vertices in X. Since $|X| \leq 3k$, it follows that there are at most $3k$ components in $G - X$. Since every component is a clique and since every vertex in a clique has the same neighborhood, every component of $G - X$ is a twin class, hence there are at most $3k(2k + 3)$ vertices in $G - X$. It follows that there are at most $6k^2 + 6k$ vertices.

Intuitively, the "real kernel" is the set X. This set contains the main parts of the graph that need to be fixed in order to obtain a cluster graph (or other graph classes when other problems are at hand). The rest of the instance, $G - X$ serves almost exclusively as a way of annotating the vertices of the modulator to which vertices they belong.

Finally, note that there is a much better—and faster— kernel for Cluster Deletion, the above exposition is for explanatory purposes only.

Chapter 4

Threshold graphs

In this chapter, we show that the edge modification problems relating to threshold graphs all have quadratic vertex kernels. This answers a recent question by Liu, Wang, and Guo [LWG14]. Since the class of threshold graphs is closed under taking complements, it follows that for every instance (G, k) of Threshold Completion, $(\overline{G}, k)$ is an equivalent instance of Threshold Deletion and vice versa by Fact 2.7. The same trick almost applies to Chain Deletion as well. Due to this, we restrict our attention to the completion and editing variants.

Threshold Editing parameterized by k

Input:	A graph G and an integer k
Question:	Does there exist a set F of at most k edges such that $G \triangle F$ is a threshold graph?

We study all three graph modification problems towards threshold graphs. The definition of the two other problems, the completion and deletion variant are exactly the same, but replace $\triangle$ with either adding the edges or deleting the edges. We start by describing the obstructions of threshold and chain graphs.

Motivated by the characterization of threshold graphs from Chapter 2, more specifically Propositions 2.21 and 2.31, we define obstructions.

Definition 4.1 (Obstruction)**.** A graph H is a *threshold obstruction* if it is isomorphic to a member of the set $\{2K_2, C_4, P_4\}$ and a *chain obstruction* if it is isomorphic to a member of the set $\{2K_2, C_3, C_5\}$. If it is clear from the context, we will often use the term obstruction for both threshold and chain obstructions and denote the set of obstructions by $\mathcal{H}$. Furthermore, if an obstruction H is an induced subgraph of a graph G we call H an *obstruction in* G.

Definition 4.2 (Realization)**.** For a graph G and a set of vertices $X \subseteq V(G)$ we say that a vertex $v \in V(G) \setminus X$ is *realizing* $Y \subseteq X$ if $N_X(v) = Y$. Furthermore, we say that a set $Y \subseteq X$ is *being realized* if there is a vertex $v \in V(G) \setminus X$ such that v is realizing Y.

Before proceeding, we observe that our kernelization algorithms does not modify any edges, and only changes the budget in the case that we discover that we have a no-instance (in which case we return $(H, 0)$, where H is an obstruction in G). The only modification of the instance is vertex deletion, hence the kernelized instance is an induced subgraph of the original graph. Since the parameter is never increased, we obtain *proper kernels* (recall Definition 2.34).

4.1 Outline of the kernelization algorithm

The kernelization algorithm consists of a twin reduction rule and an irrelevant vertex rule. The twin reduction rule is based on the observation that any obstruction containing vertices from a large enough twin class will have to be handled by edges not incident to the twin class. From this observation, we may conclude that for any twin class, we may keep only a certain amount without affecting the solutions.

A key concept of the irrelevant vertex rule is what we in the previous chapter referred to as a *modulator*. A modulator is here a set of vertices X in G of linear size in k, such that for every obstruction H in G one can add and remove edges in $[X]^2$ to turn H into a non-obstruction. First, we prove that we can in polynomial time either obtain such a set X or conclude correctly that the instance is a no-instance. The observation that $G - X$ is a threshold graph will be exploited heavily and we now fix a threshold decomposition $(\mathcal{C}, \mathcal{I})$ of $G - X$. We then prove that the idea of Proposition 2.20 can be extended to vertices in $G - X$ when considering their neighborhoods in G. In other words, the neighborhoods of the vertices in $G - X$ are nested also when considering G. This immediately yields that the number of subsets of X that are being realized is bounded linearly in the size of X and hence also in k.

We now either conclude that the graph is small or we identify a sequence of levels in the threshold decomposition containing many vertices, such that all the clique vertices and all the independent set vertices in the sequence have identical neighborhoods in X, respectively. The crux is that in the middle of such a sequence there will be a vertex that is replaceable by other vertices in every obstruction and hence is irrelevant. Such a sequence is obtained by discarding all levels in the decomposition that are extremal with respect to a subset Y of X, meaning that there either are no levels above or underneath that contain vertices realizing Y. One can prove that in this process, only a quadratic number of vertices are discarded and from this we obtain a kernel.

4.2 The twin reduction rule

First, we introduce the twin reduction rule as described above. For the remainder of the section we will assume this rule to be applied exhaustively and hence we can assume all twin classes to be small. Recall now that $\mathrm{tc}(v)$ denotes the twin class of a vertex v.

Rule 4.1 (Twin reduction rule)**.** *Let (G, k) be an instance of* Threshold Completion *or* Threshold Editing *and v a vertex in G such that $|\operatorname{tc}(v)| > 2k + 2$. We then reduce the instance to $(G - v, k)$.*

Lemma 4.3. *Let G be a graph and v a vertex in G such that $|\operatorname{tc}(v)| > 2k+2$. Then for every k we have that (G, k) is a yes-instance of* Threshold Completion *(or* Threshold Editing*) if and only if $(G - v, k)$ is a yes-instance of* Threshold Completion *(resp.* Threshold Editing*). Furthermore, we can exhaustively perform Rule 4.1 in polynomial time.*

Proof. For readability we only consider Threshold Editing, however exactly the same proof works for Threshold Completion. Let $G' = G - v$. It trivially holds that if (G, k) is a yes-instance, then also (G', k) is a yes-instance. This is due to the fact that removing a vertex never will create new obstructions.

Now, let (G', k) be a yes-instance and assume for a contradiction that (G, k) is a no-instance. Let F be an optimal solution of (G', k) and W an obstruction in $G \triangle F$. Since W is not an obstruction in G' it follows immediately that v is in W. Furthermore, since $|F| \leq k$ it follows that there are two vertices $a, b \in \operatorname{tc}(v) \setminus \{v\}$ that F is not incident with. One may observe that no obstruction contains more than two vertices from a twin class and hence we can assume without loss of generality that b is not in W. It follows that $N_{G \triangle F}(v) \cap (W - v) = N_G(v) \cap (W - v) = N_G(b) \cap (W - v) = N_{G'}(b) \cap (W - v)$ and hence the graph induced on $V(W) \triangle \{b, v\}$ is an obstruction in $G' \triangle F$, contradicting that F is a solution for (G', k).

Finally, to observe that the rule can be exhaustively applied in polynomial time, observe that we apply is at most n times, each time finding the twin class of the vertices, which is clearly doable in polynomial time. □

4.3 The modulator

Recall now that a set X is a modulator of a graph G if for every obstruction W in G we can edit edges between vertices in X to turn W into a non-obstruction. The kernelization algorithm will heavily depend on finding a small modulator X and the fact that $G - X$ is a threshold graph.

Lemma 4.4. *There is a polynomial time algorithm that given a graph G and an integer k either*

- *outputs a modulator X of G such that $|X| \leq 4k$ or*
- *correctly concludes that (G, k) is a no-instance of both* Threshold Completion *and* Threshold Editing*.*

Proof. Let X_1 be the empty set and $\mathcal{W} = \{W_1, \ldots, W_t\}$ the set of all obstructions in G. We execute the following procedure for every W_i in $\mathcal{W}$: If $W_i \triangle F$ is an obstruction for every $F \subseteq [X_i \cap V(W_i)]^2$ we let $X_{i+1} = X_i \cup V(W_i)$, otherwise we let $X_{i+1} = X_i$. After we have considered all obstructions we let $X = X_{t+1}$. If $|X| > 4k$ we conclude that (G, k) is a no-instance, otherwise we output X.

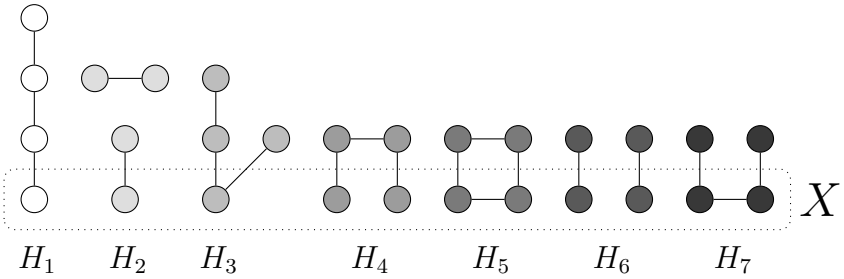


Figure 4.1: Some of the intersections with a modulator X that will not occur by definition. More specifically the ones necessary for the proof of the kernel.

Since all there is a fixed number of obstructions, the algorithm described clearly runs in polynomial time. We now argue that X is a modulator of G. If W_i was added to X_{i+1}, we let F be all the non-edges of W. Since $W \triangle F$ is isomorphic to K_4 it follows immediately that $W \triangle F$ is not an obstruction. If W_i was not added to X_{i+1}, let F the set found in $[X_i \cap V(W_i)]^2$ such that $W_i \triangle F$ is not an obstruction. Observe that $F \subseteq [X]^2$ and hence X is a modulator.

It remains to prove that if $|X| > 4k$ then (G, k) is a no-instance of Threshold Editing. Observe that it will follow immediately that (G, k) is a no-instance of Threshold Completion. Since every obstruction consists of four vertices there was at least $k+1$ obstructions added during the procedure. Assume without loss of generality that $W_1, \ldots, W_{k+1}$ was added. Observe that by construction, a solution must contain an edge in $[X_{i+1} - X_i]^2$ for every $i \in [k+1]$ and hence contains at least $k+1$ edges. □

4.4 Obtaining structure

We now study the modulator's interaction with the remaining graph in order to obtain structure of the neighborhoods between the modulator and the remaining graph. First, we prove that the neighborhoods of the vertices outside of X are nested and that the number of realized sets in X are bounded linearly in k.

Lemma 4.5. *Let G be a graph and X a modulator. For every pair of vertices u and v in $G - X$ it holds that either $N(u) \subset N[v]$ or $N(v) \subset N[u]$.*

Proof. Assume otherwise for a contradiction and let u' be a vertex in $N(u) \setminus N[v]$ and v' a vertex in $N(v) \setminus N[u]$. Let $W = G[\{u, v, u', v'\}]$ and observe that uu' and vv' are edges in W and uv' and vu' are non-edges in W by definition. Hence, no matter if some of the edges uv and $u'v'$ are present or not, W is an obstruction in G (see Figure 4.1 for an illustration). Since $u'v'$ is the only pair in W possibly with both elements in X this contradicts X being a modulator. □

Lemma 4.6. *Let G be a graph and X a modulator, then the number of X-neighborhoods,*

$$|\{N_X(v) \text{ for } v \in V(G) \setminus X\}| \leq |X| + 1.$$

Or in other words, there are at most $|X|+1$ subsets of X that are being realized.

Proof. Let u and v be two vertices in $G-X$. It follows directly from Lemma 4.5 that either $N_X(v) \subseteq N_X(u)$ or $N_X(v) \supseteq N_X(u)$. The result follows immediately. □

With the definition of the modulator and the basic properties above, we are now ready to extract more vertices from the instance, aiming at many consecutive levels that have the same neighborhood in X for the clique, and independent set vertices, respectively. This will lead up to our irrelevant vertex rule.

Let G be a graph, X a modulator and $(\mathcal{C}, \mathcal{I})$ a threshold partition of $G-X$. Recall from Definition 2.28 that $\mathcal{C} = \langle C_1, \ldots, C_t \rangle$ and $\mathcal{I} = \langle I_1, \ldots, I_t \rangle$. Letting P denote either C or I, and $P_i = C_i$ or $P_i = I_i$, we say that a subset $Y \subseteq X$ has its *upper extreme* in P_i if P_i realizes Y and for every $j > i$ it holds that P_j does not realize Y. Similarly, a subset $Y \subseteq X$ has its *lower extreme* in P_i if P_i realizes Y and for every $j < i$ it holds that P_j does not realize Y. We say that $Y \subseteq X$ is *extremal* in P_i if Y has its upper or lower extreme in Y. Observe that every $Y \subseteq X$ is extremal in at most two clique fragments and two independent set fragments.

We continue having P denote either C or I.

Lemma 4.7. *Let G be a graph, X a modulator and $(\mathcal{C}, \mathcal{I})$ a threshold partition of $G-X$. For every $Y \subseteq X$ it holds that if Y has its lower extreme in P_ℓ and upper extreme in P_u, then for every vertex $v \in P_i$ with $i \in [\ell+1, u-1]$ it holds that $N_X(v) = Y$.*

Proof. Let Y be a subset of X with C_ℓ and C_u being its lower and upper extremes in the clique respectively. By definition there is a vertex $u \in C_\ell$ and a vertex $w \in C_u$ such that $N_X(u) = N_X(w) = Y$. Let i be an integer in $[\ell+1, u-1]$ and a vertex $v \in C_i$. By the definition of a threshold partition it holds that $N_{G-X}(w) \subset N_{G-X}(v) \subset N_{G-X}(u)$. It follows from Lemma 4.5 that $N(w) \subset N[v]$ and that $N(v) \subset N[u]$. Hence,

$$Y = N_X(w) \subseteq N_X(v) \subseteq N_X(u) = Y$$

and we conclude that $N_X(v) = Y$. Since i and v was arbitrary, the proof is complete. □

We will now distinguish between three types of vertices in $G-X$. This distinction into three categories will be helpful when analyzing the number of vertices a yes-instance can have in $G-X$. It will also be useful when adding one final reduction rule, an irrelevant vertex rule (Rule 4.2).

Definition 4.8 (Important, outlying, and regular)**.** We say that P_i in the partition is *important* if there is a $Y \subseteq X$ such that Y has its extreme in P_i. Furthermore, a level L_i is important if C_i or I_i is important. Let f be the smallest number such that $|\cup_{i \leq f} C_i| \geq 2k+2$ and r the largest number such that $|\cup_{i \geq r} I_i| \geq 2k+2$. A level L_i is *outlying* if $i \leq f$ or $i \geq r$. All other levels of the decomposition are *regular* and a vertex is regular, outlying or important depending on the type of the level it is contained in.

With this characterization, we proceed to analyze the number of important levels and, more importantly, the number of important vertices in total in $G - X$. We will see that this is already bounded by $O(k^2)$.

Lemma 4.9. *Let G be a graph and X a modulator of G of size at most $4k$. Then every threshold partition of $G - X$ has at most $16k + 4$ important levels.*

Proof. The result follows immediately from the definition of important levels and Lemma 4.6. □

Lemma 4.10. *Let G be a graph, X a modulator of G and $(\mathcal{C}, \mathcal{I})$ a threshold partition of $G - X$, then for every set $Y \subseteq X$ there are at most two important clique fragments (independent fragments) realizing Y.*

We henceforth refer to C and I as the vertices comprising $\mathcal{C}$, the clique partition, and $\mathcal{I}$, the independent set partition, respectively.

Proof of Lemma 4.10. We first prove the statement for clique fragments. Let Y be a subset of X and $i < j < k$ three integers. Assume for a contradiction that C_i, C_j and C_k are important clique fragments all realizing Y. By definition there are vertices $u \in C_i$, $v \in C_j$ and $w \in C_k$ such that $N_X(u) = N_X(v) = N_X(w) = Y$. Furthermore, there is a vertex $v' \in C_j$ such that $N_X(v') \neq Y$ since C_j is important and Y does not have an extreme in C_j. By the definition of threshold partitions, we have that $N_{G-X}(w) \subset N_{G-X}(v') \subset N_{G-X}(u)$. Lemma 4.5 immediately implies that $N(w) \subset N[v']$ and $N(v') \subset N[u]$ and since $\{u, v', w\} \subseteq C$ it holds that $N[u] \subseteq N[v'] \subseteq N[w]$. Since $N_X(v') \neq Y$, we have $N_X(w) \subset N_X(v') \subset N_X(u)$, which contradicts the definition of w and u since $N_X(u) = N_X(w)$. By a symmetric argument, the statement also holds for independent fragments. □

Lemma 4.11. *Let G be a graph, X a modulator of G of size at most $4k$ and $(\mathcal{C}, \mathcal{I})$ a threshold partition of $G - X$. Then there are at most $64k^2 + 80k + 16$ important vertices in $G - X$.*

Proof. Let Y be the set of all vertices contained in a important clique or independent fragment and let Z be the set of all important vertices. Observe that $Y \subseteq Z$ and that every C_i or I_i contained in $Z \setminus Y$ is a twin class in G by definition. By Lemma 4.9 there are at most $16k + 4$ important levels and since the twin-rule has been applied exhaustively it holds that $|Z \setminus Y| \leq (16k+4)(2k+2) = 32k^2+40k+8$.

Let A be a subset of X and B the vertices in Y such that their neighborhood in X is exactly A. Let D be a C_i or I_i contained in Y and observe that $D \cap B$ is a twin class in G and hence $|D \cap B| \leq 2k+2$. And hence it follows from Lemma 4.10 that $|B| \leq 8k + 8$. Furthermore, we know from Lemma 4.6 that there are at most $4k + 1$ realized in X and hence $|Y| \leq (8k + 8)(4k + 1) = 32k^2 + 40k + 8$. It follows immediately that $|Z| \leq 64k^2 + 80k + 16$, completing the proof. □

Having bounded the number of important vertices, we proceed to bound the number of outlying vertices, also by $O(k^2)$. When this is done, what remains is to bound the number of regular vertices. However, there might still be arbitrarily

many regular vertices, and so the next section will introduce a new rule to take care of this.

Lemma 4.12. *Let G be a graph, X a modulator of G of size at most $4k$ and $(\mathcal{C}, \mathcal{I})$ a threshold partition of $G - X$. Then there are at most $80k^2 + 112k + 32$ important and outlying vertices in total in $G - X$.*

Proof. By Lemma 4.11 it follows that there are at most $64k^2 + 80k + 16$ vertices that are important and possibly outlying. It follows from Lemma 4.7 that if a level is not important its vertices are covered by at most two twin classes in G and hence the level contains at most $4k + 4$ vertices. By definition there are at most $4k+4$ outlying levels and hence at most $(4k+4)(4k+4) = 16k^2+32k+16$ vertices which are outlying, but not important. The result follows immediately. □

For the next section, where we introduce the last rule and wrap up, the following lemma will be useful. It essentially says that in any solution to the instance (G, k), no regular vertex will change from the clique partition to the independent set partition and vice versa.

Lemma 4.13. *Let G be a graph, X a modulator of G, v a regular vertex in some threshold partition $(\mathcal{C}, \mathcal{I})$ of $G - X$. Then for every $F \subseteq [V(G)]^2$ such that $G \triangle F$ is a threshold graph, $|F| \leq k$ and every split partition (C_F, I_F) of $G \triangle F$ we have:*

- *$v \in C$ if and only if $v \in C_F$ and*
- *$v \in I$ if and only if $v \in I_F$.*

Proof. Observe that the two statements are equivalent and that it is sufficient to prove the forward direction of both statements. First, we prove that $v \in C$ implies that $v \in C_F$. Let Y be the set of outlying vertices in $I \cap N_G(v)$ and recall that $|Y| > 2k+1$ by definition. Observe that at most $2k$ vertices in Y are incident to F and hence there are two vertices u, u' in Y that are untouched by F. Clearly, u and u' are not adjacent in $G \triangle F$ and hence we can assume without loss of generality that u is in I_F. Since u is untouched by F, v is adjacent to u by the definition of outlying vertices and hence v is not in I_F. A symmetric argument gives that $v \in I$ implies that $v \in I_F$ and hence our argument is complete. □

4.5 The irrelevant vertex rule

We have now obtained the structure necessary to give our irrelevant vertex rule. But before stating the rule, we need to define these consecutive levels with similar neighborhood and what it means for a vertex to be in the middle of such a collection of levels.

Definition 4.14 (Strips and centrality)**.** Let G be a graph, X a modulator and $(\mathcal{C}, \mathcal{I})$ a threshold partition of $G - X$. A *strip* is a maximal set of consecutive levels which are all regular and we say that a strip is *large* if it contains at least

$16k + 13$ vertices. For a strip $S = ([C_a, I_a], \ldots, [C_b, I_b])$ a vertex $v \in C_i$ is *central* if $a \leq i \leq b$ and $|\cup_{j\in[a,i-1]} C_j| \geq 2k + 2$ and $|\cup_{j\in[i+1,b]} C_j| \geq 2k + 2$. Similarly we say that a vertex $v \in I_i$ is *central* if $a \leq i \leq b$ and $|\cup_{j\in[a,i-1]} I_j| \geq 2k + 2$ and $|\cup_{j\in[i+1,b]} I_j| \geq 2k + 2$. Furthermore, we say that a vertex v is *central in* G if there exists a modulator X of size at most $4k$ and a threshold decomposition of $G - X$ such that v is central in a large strip.

The following lemma explicitly states the relationship between being a large strip and that of a central vertex:

Lemma 4.15. *If a strip is large it has a central vertex.*

Proof. Let $S = ([C_a, I_a], \ldots, [C_b, I_b])$ be a large strip. First, we consider the case when $|\cup_{i\in[a,b]} C_i| \geq |\cup_{i\in[a,b]} I_i|$. Observe that $|\cup_{i\in[a,b]} C_i| \geq 8k + 7$. Let i be the smallest number such that $|\cup_{j\in[a,i-1]} C_j| \geq 2k + 2$. It follows immediately from $|C_{i-1}| \leq 2k + 2$ that $|\cup_{j\in[a,i-1]} C_j| \leq 4k + 3$. Furthermore, since $|C_i| \leq 2k + 2$ it follows that $|\cup_{j\in[i+1,b]} C_j| \geq 8k + 7 - (2k + 2 + 4k + 3) = 2k + 2$. And hence any vertex in C_i is central. A symmetric argument for the case $|\cup_{i\in[a,b]} C_i| < |\cup_{i\in[a,b]} I_i|$ completes the proof. □

We are ready for the second and final rule of this kernel. Intuitively, whereas the twin reduction rule (Rule 4.1) ensured that each level contained few vertices, this rule ensures that there are few "similar-looking" levels, levels only differing in the neighborhood in $G - X$:

Rule 4.2 (Irrelevant vertex rule)**.** *If (G, k) is an instance of either* Threshold Completion *or* Threshold Editing *and v is a central vertex in G then we continue with the instance $(G - v, k)$.*

In now only remains to show that this rule is sound. That done, a very simple summary will conclude that the size of a yes-instance has at most $O(k^2)$ vertices.

Lemma 4.16. *Let (G, k) be an instance, X a modulator and v a central vertex in G. Then (G, k) is a yes-instance of* Threshold Editing *(*Threshold Completion*) if and only if $(G - v, k)$ is a yes-instance.*

Recall from Definition 2.28 that $\text{lev}(v)$ for a vertex v of a graph G with threshold partition $(\mathcal{C}, \mathcal{I})$ is the number i such that $v \in C_i \cup I_i$, that is, the index of the level in which v lives.

Proof of Lemma 4.16. For readability we only consider Threshold Editing, however the exact same proof works for Threshold Completion. For the forwards direction, for any vertex v, if (G, k) is a yes-instance, then $(G - v, k)$ is also a yes-instance. This holds since threshold graphs are hereditary.

For the reverse direction, let $(G - v, k)$ be a yes-instance and assume for a contradiction that (G, k) is a no-instance. Let F be a solution of $(G - v, k)$ satisfying Lemma 2.30, and let $G' = G \triangle F$. By assumption, (G, k) is a no-instance, so specifically, G' is not a threshold graph. Let W be an obstruction

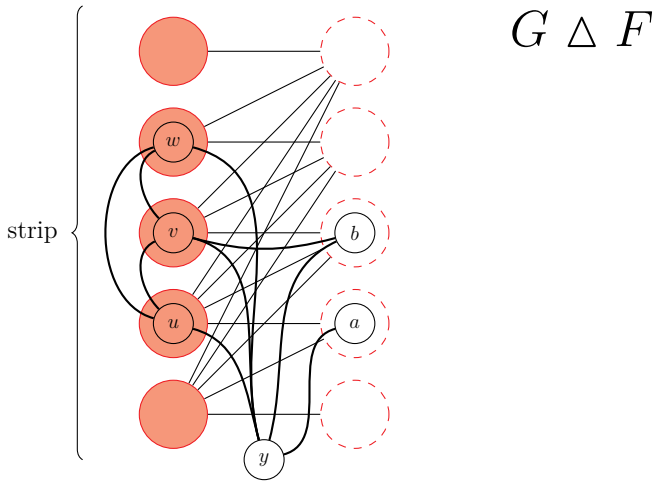


Figure 4.2: The vertex v is a center vertex in a strip and $W = \{v, a, b, y\}$ is assumed to be an obstruction.

in G'. Clearly $v \in W$ since otherwise there is an obstruction in $(G - v) \triangle F$, so consider $Z = V(W) \setminus v$. For convenience we will use N' to denote neighborhoods in G' and specifically for any set $Y \subseteq V(G')$, $N'_Y(v) = N_{G'}(v) \cap Y$. Furthermore, let $(\mathcal{C}, \mathcal{I})$ be a threshold decomposition of $G - X$ such that there is a large strip S for which v is central. We will now consider the case when v is in the clique of $G - X$. Since $|F| \leq k$ and S is a large strip it follows immediately that there are two clique vertices w and w' in S in higher levels than v that is not incident to F. Observe that $\{w, w', v\}$ forms a triangle and that W contains no such subgraph. Hence, we can assume without loss of generality that $w \notin V(W)$. Similarly, we obtain a clique vertex u in a lower level than v in S such that $u \notin W$.

Observe that $G'[Z \cup \{u\}]$ is not an obstruction and hence $N_Z(u) = N'_Z(u) \neq N'_Z(v) = N_Z(v)$. Since u and v are clique vertices from the same strip it is true that $N_X(v) = N_X(u)$ and hence there is an independent vertex a in Z such that $\text{lev}(u) \leq \text{lev}(a) < \text{lev}(v)$ (see Definition 2.28). In other words u is adjacent to a while v and w are not. By a symmetric argument we obtain a vertex b such that $\text{lev}(v) \leq \text{lev}(b) < \text{lev}(w)$, meaning that both u and v are adjacent to b while w is not. Let y be last vertex of Z, meaning that $\{v, y, a, b\} = V(W)$. Observe that a and b are regular vertices and hence it follows from Lemma 4.13 that for every threshold partition of G' it holds that $\{a, b\}$ are independent vertices.

Recall that u, v, w, a, b are all regular and hence they are in the same partitions in G' as in $G - X$ by Lemma 4.13. Furthermore, since W is an obstruction and a is neither adjacent to v nor b in G' it holds that y and a are adjacent in G'. It follows that y is a clique vertex in G' and hence it is adjacent to both u and w in G'. Since u and w are not incident to F by definition, they are adjacent to y also in G. Since u, v, w are regular and from the same strip it follows that v is adjacent to y in both G and G'. Observe that the only possible adjacency not yet decided in W

is the one between b and y. However, for W to be an obstruction it should not be present. Hence y is adjacent to a but not to b in G'. By definition $N_G(a) \subseteq N_G(b)$, however by the last observation this is not true in G'. This contradicts that F satisfies Lemma 2.30. A symmetric argument gives a contradiction for the case when v is an independent vertex and hence the proof is complete. □

The above lemma shows the soundness of the irrelevant vertex rule, Rule 4.2, and we may therefore apply it exhaustively. It should be clear that it can be exhaustively applied in polynomial time since $G - X$ is a threshold graph and we can find $(\mathcal{C}, \mathcal{I})$ in polynomial time. Marking all regular vertices and then counting successive regular levels and vertices suffices to locate an irrelevant vertex. The following theorem wraps up the goal of this section.

Theorem 1. *The following three problems admit proper kernels with at most $O(k^2)$ vertices:* Threshold Deletion, Threshold Completion *and* Threshold Editing.

Proof. Assume that Rules 4.1 and 4.2 have been applied exhaustively. If this process does not produce a modulator, we can safely output a trivial no-instance by Lemma 4.4. Hence, we can assume that we have a modulator X of size at most $4k$ and that the reduction rules cannot be applied. By Lemma 4.12 we know that there are at most $O(k^2)$ vertices in $G - X$ that are not regular. Furthermore, every regular vertex is contained in a strip and by Lemma 4.9 there are at most $O(k)$ such strips. Since the reduction rules cannot be applied, no strip is large, and hence they contain at most $O(k)$ vertices each. Since every vertex in G is either in X, or considered regular, outlying or important this gives us $O(k^2)$ vertices in total. □

Chapter 5

Chain graphs

In this chapter we provide kernels with quadratically many vertices for Chain Deletion, Chain Completion and Chain Editing. Due to the fundamental similarities between modification to chain and threshold graphs we omit the full proof and instead highlight the differences between the two proofs. Not only do we improve the current best (cubic vertex) kernel for the deletion and completion, and provide a new kernel for the editing version, but we also get rid of earlier requirements on being given a bipartite graph with fixed bipartitions as input, as was the case for the previous kernel [Guo07]. This was also the case for the subexponential parameterized time algorithm for the completion problem [FV13], and we get rid of the requirement in our algorithm in Chapter 9 as well.

Chain Editing parameterized by k

Input:	A graph G and an integer k
Question:	Does there exist a set F of at most k edges such that $G \triangle F$ is a chain graph?

We may observe that in the previous chapter, the only proofs for the threshold kernels that explicitly apply the obstructions are those of Lemmata 4.4, 4.5, and 4.16 and hence these will receive most of our attention.

The twin reduction rule goes through immediately and hence our first obstacle is the modulator. Luckily, this is a minor one. Recall from Definition 4.1 that the obstructions now are $\mathcal{H} = \{2K_2, C_3, C_5\}$; We thus get a chain-modulator X of size $5k$, as the largest obstruction contains five vertices. Besides this detail, the proof goes through exactly as it is.

5.1 An additional step

Before we continue with the remainder of the proof we need an additional step. Namely to discard all vertices that are isolated in $G - X$. We will prove that by doing this we discard at most $O(k^2)$ vertices. Now, if the irrelevant vertex rule concludes that the graph is small, then the graph is small also when we reintroduce

the discarded vertices. And if we find an irrelevant vertex, we remove it and reintroduce the discarded vertices before we once again apply our reduction rules. Due to the locality of our arguments, this is a valid approach.

Lemma 5.1. *For a graph G and a corresponding chain-modulator X there are at most $O(k^2)$ isolated vertices in $G - X$.*

Proof. Let I be the set of isolated vertices in $G - X$. We will prove that $\mathcal{F} = \{N_X(v) \mid v \in I\}$ is laminar (see Definition 2.10) and hence by Lemma 2.11 it holds that $|\mathcal{F}| \leq 2|X| \leq 10k$. It follows immediately, due to the twin reduction rule, that there are at most $10k(2k + 2) = O(k^2)$ independent vertices in $G - X$.

Assume for a contradiction that there are vertices u, v and w in I such that there exists $u' \in N_X(u) \setminus N_X(v)$ and $v' \in N_X(v) \setminus N_X(u)$ with $\{u', v'\} \subseteq N_X(w)$. These vertices intersect with the modulator as a variant of the forbidden H_5 in Figure 5.1 and hence we get a contradiction. □

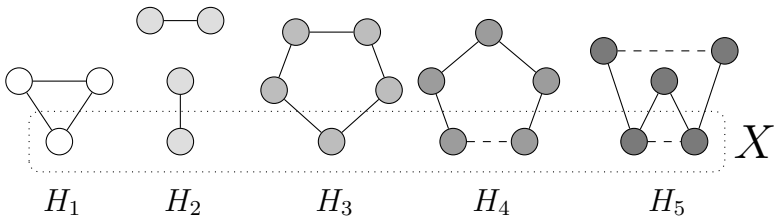


Figure 5.1: Some of the intersections of an obstruction with a chain-modulator X that by definition will not occur. Dashed edges represent edges that could or could not be there. These are the intersections necessary for the proof of the kernel.

5.2 Nested neighborhoods

From now on we will assume in all of our arguments that there are no isolated vertices in $G - X$. The next difference is with respect to Lemma 4.5, which is just not true anymore. The lemma provided us with the nested structure of the neighborhoods in the modulator and was crucial for most of the proofs. As harmful as this appears to be at first, it turns out that we can prove a weaker version that is sufficient for our needs.

Lemma 5.2 (New, weaker version of Lemma 4.5)**.** *Let G be a graph and X a chain-modulator. For every pair of vertices u and v in the* same bipartition *of $G - X$ it holds that either $N(u) \subseteq N(v)$ or $N(v) \subseteq N(u)$.*

Proof. Let u and v be two vertices from the same bipartition of $G - X$. By the definition of chain graphs we can assume that $N_{G-X}(u) \subseteq N_{G-X}(v)$. Assume for a contradiction that the lemma is not true. Then there is a vertex $u' \in N_X(u) \setminus N_X(v)$ and a vertex v' in $N_X(v) \setminus N_X(u)$. By definition, u and v are not adjacent. Since there are no isolated vertices in $G - X$ there is a vertex $a \in N_{G-X}(u) \subseteq N_{G-X}(v)$.

Observe that if a is adjacent to either u' or v' we get a C_3 that only has one vertex in X, which is a contradiction (see H_1 in Figure 5.1). However, if a is not adjacent to both u' and v' then $\{u, v, u', v', a\}$ forms the same interaction with the modulator as H_4 in Figure 5.1 and hence our proof is complete. □

One can observe that Lemma 5.2 is a sufficiently strong replacement for Lemma 4.5 since all proofs are applying the lemma to vertices from only one partition of $G - X$. The only exception is the proof of Lemma 4.6, but by applying Lemma 5.2 on one partition at the time we obtain the following bound instead:

$$|\{N_X(v) \text{ for } v \in V(G) \setminus X\}| \leq 2|X| + 2.$$

5.3 An irrelevant vertex rule

It only remains to prove that the irrelevant vertex rule can still be applied with this new set of obstructions. Although the strategy is the same, the details are different and hence we provide the proof in full detail.

Lemma 5.3. *Let (G, k) be an instance, X a modulator and v a central vertex in G. Then (G, k) is a yes-instance of* Chain Editing *(*Chain Completion*) if and only if $(G - v, k)$ is a yes-instance.*

Proof. For readability we only consider Chain Editing, however the exact same proof works for Chain Completion. For the forwards direction, for any vertex v, if (G, k) is a yes-instance, then $(G - v, k)$ is also a yes-instance. This holds since chain graphs are hereditary.

For the reverse direction, let $(G - v, k)$ be a yes-instance and assume for a contradiction that (G, k) is a no-instance. Let F be a solution of $(G - v, k)$ satisfying Lemma 2.30, and let $G' = G \triangle F$. By assumption, (G, k) is a no-instance, so specifically, G' is not a chain graph. Let W be an obstruction in G'. Clearly $v \in W$, since otherwise there is an obstruction in $(G - v) \triangle F$. Let $Z = V(W) - v$. For convenience we will use N' to denote neighborhoods in G' and specifically for any set $Y \subseteq V(G')$, $N'_Y(v) = N_{G'}(v) \cap Y$. Furthermore, let $(\mathcal{A}, \mathcal{B})$ be a chain decomposition of $G - X$ such that there is a large strip S for which v is central. Let $A = \cup\mathcal{A}$ and $B = \cup\mathcal{B}$. We will now consider the case when v is in A. Since $|F| \leq k$ and S is a large strip it follows immediately that there are two vertices w and w' in $A \cap S$ in higher levels than v that is not incident to F. Observe that $\{w, w', v\}$ forms an independent set of size three and that W contains no such subgraph. Hence, we can assume without loss of generality that $w \notin V(W)$. Similarly, we obtain a vertex u in A at a lower level than v in S such that $u \notin W$.

Observe that $G'[Z \cup \{u\}]$ is not an obstruction and hence $N_Z(u) = N'_Z(u) \neq N'_Z(v) = N_Z(v)$. Since u and v are vertices in A from the same strip it is

true that $N_X(v) = N_X(u)$ and hence there is a vertex a in $Z \cap B$ such that $\text{lev}(u) \leq \text{lev}(a) < \text{lev}(v)$. In other words u is adjacent to a, while v and w are not. By a symmetric argument we obtain a vertex b such that $\text{lev}(v) \leq \text{lev}(b) < \text{lev}(w)$, meaning that both u and v are adjacent to b while w is not. We now fix a chain decomposition $(\mathcal{A}', \mathcal{B}')$ and let $A' = \cup\mathcal{A}'$ and $B' = \cup\mathcal{B}'$. Observe that a and b are regular vertices and hence it follows from the chain version of Lemma 4.13 that $\{a, b\}$ is in B'. This yields immediately that W is not a C_3 (since a and b are not adjacent) and hence we are left the cases of W being a $2K_2$ or a C_5.

We now consider the case when W is isomorphic to a $2K_2$. Let y be the last vertex of Z, meaning that $\{v, y, a, b\} = V(W)$. Observe that since W is a $2K_2$ it holds that y is adjacent to a, but not to b. However, in G it holds that $N(a) \subseteq N(b)$ and hence F is not satisfying Lemma 2.30, which is a contradiction.

Hence we are left with the case that W is isomorphic to a C_5. Let y, x be the last vertices of Z. Observe that all vertices in W should be of degree two and hence a is adjacent to both x and y. Recall that a is in B' and observe that u is in A' by the same reasoning. Due to their adjacency to a, also x and y is in A'. It follows immediately that u, x and y form an independent set in $(G - v) \triangle F$. Since u and v are not touched by F and in the same strip it follows that v, x and y form an independent set in G'. We observe that by this W can not be isomorphic to a C_5. The argument for the case when $v \in B$ is symmetrical and hence the proof is complete. □

With the new irrelevant vertex rule and the bound on the number of possible X-neighborhoods, we obtain our kernelization results for modifications into chain graphs. The proof is exactly the same as for Theorem 1, the corresponding problems for threshold graphs.

Theorem 2. *The following three problems admit proper kernels with at most $O(k^2)$ vertices:* Chain Deletion, Chain Completion, *and* Chain Editing.

Chapter 6

Trivially perfect graphs

In this chapter we describe an $O(k^7)$ vertex kernel for Trivially Perfect Editing. This is done in the first section of this chapter. In the second section, § 6.2, we see that by modifying only two rules in total, we obtain polynomial kernels for the deletion and the completion variant as well.

Trivially Perfect Editing parameterized by k

Input:	A graph G and an integer k
Question:	Does there exist a set F of at most k edges such that $G \triangle F$ is trivially perfect?

The latter result (originally, a cubic vertex kernel for co-Trivially Perfect Deletion) was already announced at the 18th *International Symposium on Algorithms and Computation* (ISAAC) in 2007 by Guo [Guo07], but unfortunately neither the proof nor the rules of the kernel were published. We therefore add the short modifications necessary to adapt the kernel for the editing version, together with the corresponding proof, for completeness.

6.1 A kernel for editing towards trivially perfect graphs

This section is devoted to the proof of Theorem 3, stating that Trivially Perfect Editing admits a proper kernel with $O(k^7)$ vertices. But before starting the formal description, we go through a brief overview of the structure of the proof.

In Section 6.1.1 we give some preliminary basic rules, that mostly deal with situations where we can find a large number of induced C_4s and P_4s in the graph (henceforth called *obstacles*), that share only one edge or non-edge. We then infer that this edge or non-edge has to be included in any editing set of size at most k, and hence we can perform the necessary edit and decrement the budget.

In Section 6.1.2 we construct a modulator based on the characterization that trivially perfect graphs are exactly the $\{C_4, P_4\}$-free graphs. We refer to this

modulator, X, as modulator. Recall again that $|X| \leq 4k$ (otherwise, (G, k) is a no-instance) and $G - X$ is a trivially perfect graph. We will see that $G - X$ and the interaction between $G - X$ imposes a lot of structure on the considered instance, and is the key for further analysis of irrelevant parts of the input.

In Definition 2.24, we defined a *universal clique decomposition*, or *UCD* for short. A UCD is a recursive decomposition of a trivially perfect graph G into universal vertices. Recall that a graph G is trivially perfect if and only if it admits a universal clique decomposition $(T = (V_T, E_T), \mathcal{B} = \{B_t\}_{t \in V_T})$. Further, recall that for a given set of vertices $X \subseteq V(G)$, we refer to $N_X(v)$, or the X-neighborhood of v as the neighbors of v in X (Definition 2.1).

In Section 6.1.4 we proceed to analyze the trivially perfect graph $G - X$. Having the polynomial bound on the number of neighborhoods within X, we can locate in the UCD of $G - X$ a polynomial (in k) number of *important bags*, where something interesting from the point of view of X-neighborhoods happens. The parts between the important bags have very simple structure. They are either *tassels*: sets of trees hanging below some important bag, where each such tree is a module in the whole graph G; or *combs*: long paths stretched between two important bags where all the vertices of subtrees attached to the path have exactly the same neighborhood in X. Tassels and combs are treated differently: Large tassels contain large trivially perfect modules in G that can be reduced quite easily, however for combs we need to devise a quite complicated irrelevant vertex rule that locates a vertex that can be safely discarded in a long comb. The module reduction rules are described in Section 6.1.5, while in Section 6.1.6 we reduce the sizes of tassels and combs and conclude the proof.

The algorithm consists of five parts, each of which will be covered in the following sections:

1. Run set of basic rules
2. Modulator construction
3. Bounding modulator neighborhoods
4. Locating important bags
5. Module reduction
6. Kernelizing non-important parts (irrelevant vertex deletions)

6.1.1 Basic rules

In this section we introduce the first two basic reduction rules. In the argumentation of the next sections, we will assume that none of these rules is applicable. An instance satisfying this property will be called *reduced*.

See Figures 6.1 and 6.2 for illustrations of Rules 6.1 and 6.2. The red dotted edges are non-edges; They form a matching in the complement graph. In each of the cases, the only common vertices are u and v.

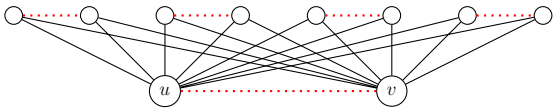


Figure 6.1: Rule 6.1: There are four C_4s sharing only the vertices u and v. Unless the edge uv is added, we must use at least as many edits as the size of the non-matching.

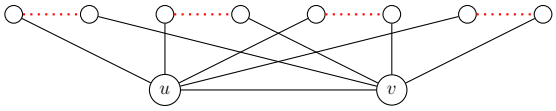


Figure 6.2: Rule 6.2: There are four P_4s sharing only the vertices u and v. Unless the edge uv is deleted, we must use at least as many edits as the size of the non-matching.

Rule 6.1. *For an instance (G, k) with $uv \notin E(G)$, if there is a matching of size at least $k+1$ in $\overline{G[N(u) \cap N(v)]}$, then add edge uv to G and decrease k by one, i.e., return the new instance $(G + uv, k-1)$.*

Rule 6.2. *For an instance (G, k) with $uv \in E(G)$ and $N_1 = N(u) \setminus N[v]$ and $N_2 = N(v) \setminus N[u]$, if there is a matching in $\overline{G}$ between N_1 and N_2 of size at least $k+1$, then delete edge uv from G and decrease k by one, i.e., return the new instance $(G - uv, k-1)$.*

Lemma 6.1. *Applicability of Rules 6.1 and 6.2 can be recognized in polynomial time. Moreover, both these rules are safe, i.e., the input instance (G, k) is a yes-instance if and only if the output instance $(G', k-1)$ is a yes-instance.*

Proof. Observe that verifying the applicability of Rule 6.1 or 6.2 to a fixed edge or non-edge uv boils down to computing the cardinality of the maximum matching in an auxiliary graph. This problem is well-known to be solvable in polynomial time [Edm65]. Thus, by iterating over all edges and non-edges of G we obtain polynomial time algorithms for recognizing applicability of Rules 6.1 and 6.2. We proceed to the proof of the safeness for both rules.

Proof of Rule 6.1. Let $x_0y_0, x_1y_1, \ldots, x_ky_k$ be the edges of the found matching in the graph $\overline{G[N(u) \cap N(v)]}$. Observe that for each i, $0 \leq i \leq k$, the vertices $\langle u, x_i, v, y_i \rangle$ induce a C_4 in G. These induced four-cycles share only the non-edge uv, hence any editing set that does not contain uv must contain at least one element of $[\{u, x_i, v, y_i\}]^2 \setminus \{uv\}$, and consequently be of size at least $k+1$. We infer that every editing set for G that has size at most k has to include the edge uv, and the safeness of the rule follows.

In plain words, Rule 6.1 states that there are $k+1$ vertex disjoint (except from u and v) cycles of length four using v and u on the diagonals. Since any editing set can touch at most k of these cycles, if we do not add the edge uv, there will remain one C_4 in any editing of size at most k. Hence adding uv is necessary, and thus we can reduce k. ⌟

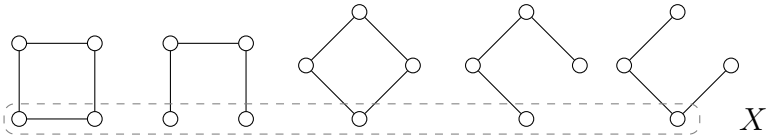


Figure 6.3: Forbidden patterns of intersection between an obstruction and a modulator X.

Proof of Rule 6.2. We proceed similarly as for Rule 6.1. Suppose that we have found a matching, $x_0y_0, x_1y_1, \ldots, x_ky_k$ in $\overline{G}$, where $x_i \in N_1$ and $y_i \in N_2$ for $0 \leq i \leq k$. Then the vertices $\langle x_i, u, v, y_i \rangle$ induce a P_4, and all these P_4s for $0 \leq i \leq k$ pairwise share only the edge uv. Similarly as for Rule 6.1, we conclude that every editing set for G of size at most k has to contain uv, and the safeness of the rule follows. ⌟

Combining the applicability of the two rules in polynomial time with their now proved soundness, the lemma follows. □

We can now use Lemma 6.1 to apply Rules 6.1 and 6.2 exhaustively; note that each application reduces the budget k, hence at most k applications can be performed before discarding the instance as a no-instance. From now on, we assume that the considered instance (G, k) is reduced.

6.1.2 Modulator construction

We now move to the construction of the modulator. Recall the $W \subseteq V(G)$ is an *obstruction* if $G[W]$ is isomorphic to C_4 or P_4:

Definition 6.2 (Modulator)**.** Let (G, k) be an instance of Trivially Perfect Editing. A subset $X \subseteq V(G)$ is a *modulator* if for every obstruction W, the following holds (see Figure 6.3):

- $|W \cap X| \geq 2$, and
- if $|W \cap X| = 2$, then it cannot happen that $G[W]$ is a C_4 or a P_4 of the form $\langle x_1, y_1, y_2, x_2 \rangle$, where $W \cap X = \{x_1, x_2\}$.

We call a modulator X *small* if $|X| \leq 4k$.

We now proceed to show exactly how to construct such a modulator, or conclude that the given instance is a no-instance. We essentially do this by a greedy packing of obstacles, and if we find more than k obstacles we may conclude that we are dealing with a no-instance.

Lemma 6.3. *Given an instance (G, k) of* Trivially Perfect Editing*, we can in polynomial time construct a small modulator $X \subseteq V(G)$, or correctly conclude that (G, k) is a no-instance.*

Proof. The algorithm starts with $X_0 = \emptyset$, and iteratively constructs an increasing family of sets $X_0 \subseteq X_1 \subseteq X_2 \subseteq \ldots$. In the ith iteration we look for an obstacle W that contradicts the fact that X_{i-1} is a modulator. If this check verifies that X_{i-1} is a modulator, then we terminate the algorithm and output $X = X_{i-1}$. Otherwise, we set $X_i = X_{i-1} \cup W$ and proceed to the next iteration. Moreover, if we performed $k+1$ iterations, i.e., successfully constructed set X_{k+1}, then we terminate the algorithm concluding that (G, k) is a no-instance. Since in each iteration the next X_i grows by at most four vertices, we infer that if we succeed in outputting a modulator X, then it has size at most $4k$.

We are left with proving that if the algorithm successfully constructed X_{k+1}, then (G, k) is a no-instance. To this end, we prove by induction on i that for every $i = 0, 1, \ldots, k+1$ and every editing set F for G, it holds that $|F \cap [X_i]^2| \geq i$. Indeed, from this statement for $i = k+1$ we can infer that every editing set for G has size at least $k+1$, so (G, k) is a no-instance. The base of the induction is trivial, so for the induction step suppose that $X_i = X_{i-1} \cup W$, where W is an obstacle with $|W \cap X_{i-1}| \leq 1$.

First, if $|W \cap X_{i-1}| \leq 1$, then $[W]^2$ is disjoint with $[X_{i-1}]^2$. Since F is an editing set for G, we have that $F \cap [W]^2 \neq \emptyset$, and hence

$$\left|F \cap [X_i]^2\right| \geq \left|F \cap [X_{i-1}]^2\right| + \left|F \cap [W]^2\right| \geq i - 1 + 1 = i,$$

by the induction hypothesis. Second, if $|W \cap X_{i-1}| = 2$ and W has one of the two forms described in the second point of Definition 6.2, then it is easy to see that F in fact has to have a nonempty intersection with $[W]^2 \setminus \{x_1x_2\}$: editing only the (non)edge x_1x_2 would turn a C_4 into a P_4 or vice versa. Since $[W]^2 \setminus \{x_1x_2\}$ is disjoint with $[X_{i-1}]^2$, we analogously obtain that

$$\left|F \cap [X_i]^2\right| \geq \left|F \cap [X_{i-1}]^2\right| + \left|F \cap \left([W]^2 \setminus \{x_1x_2\}\right)\right| \geq i - 1 + 1 = i.$$

□

6.1.3 Bounding the number of modulator neighborhoods

We now compute the universal clique decomposition $\mathcal{T} = (T, \mathcal{B})$ of the trivially perfect graph $G - X$. The goal of this section is to analyze the structure of neighborhoods within X of vertices residing outside X.

Extending the quasi-order $\preceq$ on $G - X$. Again, we shall omit the subscript G whenever this does not lead to any confusion. Recall that the UCD $\mathcal{T}$ gives us a quasi-ordering $\preceq$ on the vertices of $G - X$. We have $u \preceq v$ if the bag to which v belong is a descendant of the bag which u belongs to, where every bag is considered its own descendant. We shall use the notation $u \prec v$ to denote that $u \preceq v$ and $v \not\preceq u$. The following two lemmata show that the quasi-ordering $\preceq$ is compatible with the inclusion ordering of X-neighborhoods.

Lemma 6.4. *If $u \prec v$ then $N_X(u) \supseteq N_X(v)$.*

Proof. Suppose $u \in B_t$ and $v \in B_s$, where $t \neq s$ and t is an ancestor of s in the forest T. Recall that in a UCD, every non-leaf node has at least two children, which means that there exists some node s' that is a descendant of t, but which is incomparable with s. Let w be any vertex of $B_{s'}$. From the definition of a UCD it follows that $uv, uw \in E(G)$ but $vw \notin E(G)$.

For the sake of contradiction suppose that $N_X(u) \not\supseteq N_X(v)$, which means there exists a vertex $x \in X$ with $xv \in E(G)$ and $xu \notin E(G)$. It follows that $\{x, u, v, w\}$ is an obstacle regardless of whether wx is an edge or a non-edge: it is an induced C_4 if $wx \in E(G)$ and an induced P_4 if $wx \notin E(G)$. Thus we have uncovered an obstacle sharing only one vertex with X, contradicting the fact that X is a modulator. □

Lemma 6.5. *If $u, v \in B_t$ for some $B_t \in \mathcal{B}$, then $N_X(u) \subseteq N_X(v)$ or $N_X(v) \subseteq N_X(u)$.*

Proof. Since $u, v \in B_t$, we have that $uv \in E(G)$. For the sake of contradiction, suppose that there exist some $x_u \in N_X(u) \setminus N_X(v)$ and $x_v \in N_X(v) \setminus N_X(u)$. It can be now easily seen that regardless whether $x_u x_v$ belongs to $E(G)$ or not, the quadruple $\{u, v, x_u, x_v\}$ forms one of the obstacles forbidden in the second point of the Definition 6.2. This is a contradiction with the fact that X is a modulator. □

Lemmata 6.4 and 6.5 motivate the following refinement of the quasi-ordering $\preceq$: If u, v belong to different bags of $\mathcal{T}$, then we put $u \preceq_N v$ if and only if $u \preceq v$, and if they are in the same bag, then $u \preceq_N v$ if and only if $N_X(u) \supseteq N_X(v)$. Thus, by Lemma 6.5 $\preceq_N$ refines $\preceq$ by possibly splitting every bag of $\mathcal{T}$ into a family of linearly ordered equivalence classes. Moreover, by Lemmata 6.4 and 6.5 we have the following corollary.

Corollary 6.6. *If $u \preceq_N v$ then $N_X(u) \supseteq N_X(v)$.*

Observe that for a pair of vertices $u, v \in V(G) \setminus X$, the following conditions are equivalent: *(a)* u and v are comparable with respect to $\preceq$, *(b)* u and v are comparable with respect to $\preceq_N$, and (c) $uv \in E(G)$. We have now prepared all the tools needed to prove the main lemma from this section. For the proof of the lemma we will use the fact that a weakly laminar set system has size at most the size of the ground set plus one. See Definition 2.12 for the definition of a weakly laminar set system, and Lemma 2.13 for the size bound.

Lemma 6.7. *If (G, k) is a reduced instance for* Trivially Perfect Editing *and X is a small modulator, then the number of different X-neighborhoods is at most $O(k^4)$.*

Proof. Let $\mathcal{F}$ be the family of X-neighborhoods in G. For every $Z \in \mathcal{F}$, let us choose an arbitrary vertex $v_Z \in V(G) \setminus X$ with $Z = N_X(v_Z)$. We split $\mathcal{F}$ into two subfamilies: The first family $\mathcal{F}_1$ contains all the sets of $\mathcal{F}$ that contain the endpoints of some non-edge in $G[X]$, whereas the second family $\mathcal{F}_2$ contains all the sets of $\mathcal{F}$ that induce complete graphs in $G[X]$. We bound the sizes of $\mathcal{F}_1$ and $\mathcal{F}_2$ separately.

Bounding $|\mathcal{F}_1|$: Let xy be a non-edge of $G[X]$, and for $2 \leq \kappa \leq |X|$ let $\mathcal{F}_1^{xy,\kappa}$ be the family of those sets of $\mathcal{F}_1$ that contain $\{x, y\}$ and have cardinality exactly κ. Take any distinct $Z_1, Z_2 \in \mathcal{F}_1^{xy,\kappa}$, and observe that they are not nested since both have size κ. By Corollary 6.6, this means that vertices v_{Z_1} and v_{Z_2} are incomparable with respect to $\preceq_N$, so $v_{Z_1}v_{Z_2} \notin E(G)$. Hence, set $\{v_Z \colon Z \in \mathcal{F}_1^{xy,\kappa}\}$ is independent in G. Observe now that if we had that $|\{v_Z \colon Z \in \mathcal{F}_1^{xy,\kappa}\}| \geq 2k+2$, then Rule 6.1 would be applicable to the non-edge xy. Since we assume that the instance is reduced, we conclude that $|\{v_Z \colon Z \in \mathcal{F}_1^{xy,\kappa}\}| \leq 2k+1$, and hence also $|\mathcal{F}_1^{xy,\kappa}| \leq 2k+1$. By summing through all the κ between 2 and $|X|$ and through all the non-edges of $G[X]$, we infer that

$$|\mathcal{F}_1| \leq \binom{4k}{2} \cdot 4k \cdot (2k+1) = O(k^4).$$

Bounding $|\mathcal{F}_2|$: Consider any pair of X-neighborhoods $Z_1, Z_2 \in \mathcal{F}_2$ such that they are not nested, and moreover there exist vertices $x_1 \in Z_1 \setminus Z_2$ and $x_2 \in Z_2 \setminus Z_1$ such that $x_1x_2 \in E(G)$. Since Z_1 and Z_2 are not nested, by Corollary 6.6 we infer that v_{Z_1} and v_{Z_2} are incomparable with respect to $\preceq_N$, and hence $v_{Z_1}v_{Z_2} \notin E(G)$. Observe that then $G[\{v_{Z_1}, v_{Z_2}, x_1, x_2\}]$ is an induced P_4; however, the existence of such an obstacle is not forbidden by the definition of a modulator.

Create an auxiliary graph H with $V(H) = \mathcal{F}_2$, and put $Z_1Z_2 \in E(H)$ if and only if Z_1 and Z_2 satisfy the condition from the previous paragraph, i.e., Z_1 and Z_2 are not nested and there exist $x_1 \in Z_1 \setminus Z_2$ and $x_2 \in Z_2 \setminus Z_1$ with $x_1x_2 \in E(G)$. Run the classic greedy 2-approximation algorithm for vertex cover in H. This algorithm either finds a matching M in H of size more than $\binom{4k}{2} \cdot k$, or a vertex cover C of H of size at most $2 \cdot \binom{4k}{2} \cdot k$. In the first case, assign each edge Z_1Z_2 of M to the corresponding edge x_1x_2 of $G[X]$ as in the definition of the edges of H. Observe that since $|X| \leq 4k$, then some edge $x_1x_2 \in G[X]$ is assigned at least $k+1$ times. Then it is easy to see that the sets $\{v_{Z_1}, v_{Z_2}, x_1, x_2\}$ for Z_1Z_2 being edges of M assigned to x_1x_2 induce P_4s that share only the edge x_1x_2, and hence Rule 6.2 would be applicable to x_1x_2. This is a contradiction with the assumption that (G, k) is reduced. Hence, we can assume that we have successfully constructed a vertex cover C of H of size at most $2 \cdot \binom{4k}{2} \cdot k = O(k^3)$.

Let now $\mathcal{F}_2' = \mathcal{F}_2 \setminus C$. Since $\mathcal{F}_2'$ is independent in H, it follows that for any non-nested $Z_1, Z_2 \in \mathcal{F}_2'$ and any $x_1 \in Z_1 \setminus Z_2$, $x_2 \in Z_2 \setminus Z_1$, we have that $x_1x_2 \notin E(G)$. Since the sets of $\mathcal{F}_2'$ induce complete graphs in $G[X]$, this means that in particular there is no set $Z_3 \in \mathcal{F}_2'$ that contains both x_1 and x_2. This proves that the family $\mathcal{F}_2'$ is a weakly laminar set system with X as ground set, so by Lemma 2.13 we infer that $|\mathcal{F}_2'| \leq |X| + 1 \leq 4k+1$. Concluding,

$$|\mathcal{F}_2| \leq |C| + |\mathcal{F}_2'| \leq O(k^3) + 4k + 1 = O(k^3),$$

and $|\mathcal{F}| \leq |\mathcal{F}_1| + |\mathcal{F}_2| = O(k^4) + O(k^3) = O(k^4)$. □

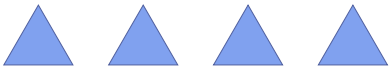


Figure 6.4: Neighborhood Type 0: x sees a disjoint union of connected components.

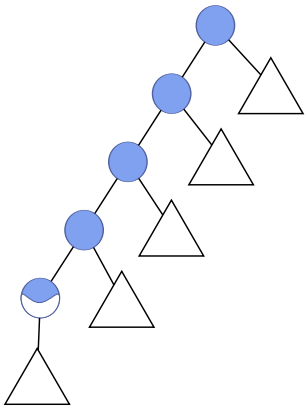


Figure 6.5: Neighborhood Type 1: x sees all the vertices in bags from a root and to a point in a bag, and nothing else.

6.1.4 Locating important bags

In the previous section we analyzed the structure of neighborhoods that vertices from $V(G) \setminus X$ have in X. Our goal in this section is to perform the symmetric analysis: to understand, how the neighborhood of a fixed $x \in X$ in $V(G) \setminus X$ looks like. Eventually, we aim to locate a family I of $O(k)$ *important bags*, where some non-trivial behavior with respect to the neighborhoods of vertices of X happens. Then, we will perform a lowest common ancestor-closure on the set I, thus increasing its size to at most twice that of its original size. After performing this step, all the connected components of $T - I$ have very simple structure from the point of view of their neighborhoods in X. As there are only $O(k)$ such components, we will be able to kernelize them separately.

The following definition and lemma explains what are the types of neighborhoods that vertices of X can have in $V(G) \setminus X$. To simplify the notation, in the following we treat $\preceq$ also as a partial order on the vertices of the forest T denoting the ancestor-descendant relation, i.e., $s \preceq t$ if and only if s is an ancestor of t (possibly $s = t$).

See Figures 6.4, 6.5, and 6.6, which depicts the three types of neighborhoods; simply denoted Type 0, Type 1, and Type 2. The blue parts in the figures mark the possible neighborhoods of a vertex $x \in X$.

Definition 6.8 (Type 0, 1, and 2 neighborhoods)**.** Let $x \in X$ be any vertex and consider $U_x = N(x) \setminus X$. We say that U_x is:

A neighborhood of Type 0
: if U_x is the union of the vertex sets of a collection of connected components of $G - X$.

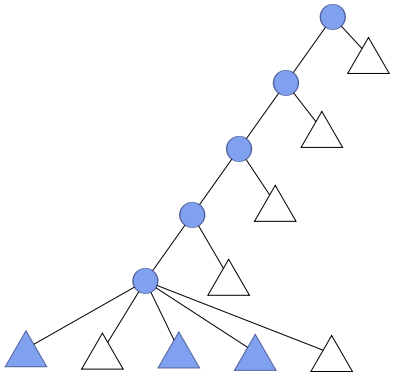


Figure 6.6: Neighborhood Type 2: x has as neighbors all the vertices from a root and down to a bag, and a collection of subtrees below that bag.

A neighborhood of Type 1
if there exists a node $t_x \in V(T)$ such that

$$\bigcup_{s \prec t_x} B_s \subseteq U_x \subseteq \bigcup_{s \preceq t_x} B_s.$$

In other words, U_x consists of all the vertices contained in bags on the path from t_x to the root of its subtree in T, where some vertices of B_{t_x} itself may be excluded.

A neighborhood of Type 2
if there exists a node $t_x \in V(T)$ and a collection $\mathcal{L}_x$ of subtrees of T rooted at children of t_x such that

$$U_x = \bigcup_{s \preceq t_x} B_s \cup \bigcup_{S \in \mathcal{L}_x} \bigcup_{s \in V(S)} B_s.$$

In other words, U_x is formed by all the vertices contained in bags on the path from t_x to the root of its subtree in T, plus a selection of subtrees rooted in the children of t_x, where the vertices appearing in the bags of each such subtree are either all included in U_x or all excluded from U_x.

The following lemma shows that the neighborhood types described above are complete, that is, that every neighborhood U_x of a vertex $x \in X$ is one of the three types above.

Lemma 6.9. *Let $x \in X$ be any vertex and consider $U_x = N(x) \setminus X$. Then U_x is of Type 0, 1 or 2.*

Proof. From Corollary 6.6 we infer that U_x is closed downwards with respect to the quasi-ordering $\preceq_N$, i.e., if $v \in U_x$ and $u \preceq_N v$, then also $u \in U_x$. Let S_x be the set of nodes of T whose bags contain at least one vertex of U_x. It follows that S_x is closed under taking ancestors in forest T. Moreover if $t \in S_x$, then the bags of all the ancestors of t other than t are fully contained in U_x.

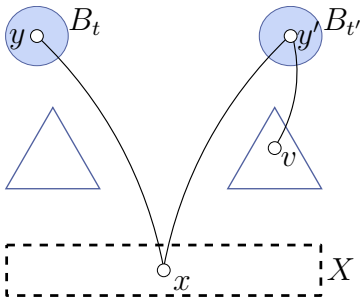


Figure 6.7: An induced P_4 on $\langle y, x, y', v\rangle$, with only one vertex, x, in the modulator, appearing in the proof of Claim 6.10.

Claim 6.10. *Suppose $t, t' \in S_x$ are two nodes that are incomparable with respect to $\preceq$. Then $U_x \supseteq \bigcup_{s \succeq t} B_s$ and $U_x \supseteq \bigcup_{s \succeq t'} B_s$, i.e., U_x contains all the vertices of all the bags contained in the subtrees of T rooted at t and t'.*

Proof of claim. We prove the statement for the subtree rooted at t'; The proof for the subtree rooted at t is symmetric. Let y and y' be arbitrary vertices of $B_t \cap U_x$ and $B_{t'} \cap U_x$, respectively. For the sake of contradiction suppose there exists some $v \in \bigcup_{s \succeq t'} B_s$ such that $vx \notin E(G)$. Since $v \in \bigcup_{s \succeq t'} B_s$ and t, t' are incomparable with respect to $\preceq$, by the properties of the universal clique decomposition we have that $yy' \notin E(G)$, $vy \notin E(G)$ and $vy' \in E(G)$. Since $xy, xy' \in E(G)$ by the definition of U_x, we conclude that $\{y, y', x, v\}$ would induce a P_4 in G that has only one vertex in common with X (see Figure 6.7), a contradiction to the definition of a modulator. ⌟

We now use Claim 6.10 to perform a case study that recognizes U_x as a neighborhood of Type 0, 1, or 2.

Suppose first that U_x contains vertices of at least two distinct connected components of $G - X$. Let C_1, C_2 be any two such components, and let T_1 and T_2 be the trees of the forest T that are UCDs of C_1 and C_2, respectively. Since S_x is closed under taking ancestors in T, it follows that the roots of T_1 and T_2 belong to S_x. Claim 6.10 implies then that the entire vertex sets of C_1 and C_2 are contained in U_x. Since C_1, C_2 was an arbitrary pair of components containing a vertex of U_x, it follows that U_x must be the union of vertex sets of a selection of connected components of $G - X$, i.e., a neighborhood of Type 0.

Since $U_x = \emptyset$ is also a neighborhood of Type 0, we are left with analyzing the case when $U_x \subseteq V(C_0)$ for C_0 being a connected component of $G - X$; Let T_0 be the UCD of C_0. Observe that if U_x does not contain any pair of vertices incomparable with respect to $\preceq$, then S_x must form a path from some node of T_0 to the root of T_0, and hence U_x is a neighborhood of Type 1. Otherwise, there exists some node of S_x such that at least two subtrees rooted at its children contain nodes from S_x. Let t_x be such a node that is highest in T_0, and let $\mathcal{L}_x$ be the family of subtrees rooted at children of t_x that contain nodes of S_x. Again applying Claim 6.10, we infer that U_x contains all the vertices of all the bags of every subtree of $\mathcal{L}_x$: for any two distinct subtrees $T_1, T_2 \in \mathcal{L}_x$, S_x contains the

roots of T_1 and T_2, and hence by Claim 6.10 U_x contains all the vertices of all the bags of T_1 and T_2. Since t_x was chosen to be the highest, it follows that U_x is a neighborhood of Type 2 for node t_x and selection of subtrees $\mathcal{L}_x$. □

Clearly, for every $x \in X$ we can in polynomial time analyze U_x and recognize it as a neighborhood of Type 0, 1, or 2. Let I_0 be the set of nodes t_x for vertices $x \in X$ for which U_x is of Type 1 or 2. To simplify the structure of $T - I_0$, we perform the lowest common ancestor-closure operation on I_0. The following variant of this operation is taken verbatim from the work of Fomin, Lokshtanov, Misra, and Saurabh on Planar $\mathcal{F}$-Deletion [FLMS12].

Definition 6.11 ([FLMS12])**.** For a rooted tree T and vertex set $M \subseteq V(T)$ the lowest common ancestor-closure (LCA-closure) is obtained by the following process. Initially, set $M' = M$. Then, as long as there are vertices x and y in M' whose least common ancestor w is not in M', add w to M'. When the process terminates, output M' as the LCA-closure of M. The following folklore lemma summarizes two basic properties of LCA-closures.

Lemma 6.12 ([FLMS12])**.** *Let T be a tree, $M \subseteq V(T)$ and $M' = \text{LCA-closure}(M)$. Then $|M'| \leq 2|M|$ and for every connected component C of $T - M'$, $|N(C)| \leq 2$.*

Construct now the set I by taking LCA-closure(I_0) and adding the root of every connected component of T that contains a bag of I_0 (provided it is not already included). The nodes from I will be called *important nodes*, or *important bags*. From Lemma 6.12 it follows that $|I| \leq 3|X| \leq 12k$, and by the construction we infer that every connected component C of $T - I$ is of one of the following three forms:

- C is not adjacent to any node of I, and is thus simply a connected component of T that does not contain any important bag.
- C is adjacent to one node a of I, and it is a subtree rooted at a child of a.
- C is adjacent to two nodes a and b of I such that a is an ancestor of b. Then C is formed by the internal nodes of the $a - b$ path in T, plus all the subtrees rooted at the other children of these internal nodes.

6.1.5 Module and twin reduction

In this section we give two new reduction rules: a twin reduction and a module reduction rule. These rules are executed exhaustively by the algorithm as Rules 6.3 and 6.4. The reason why we introduce them now is that only after understanding the structural results of Sections 6.1.3 and 6.1.4, the motivation of these rules becomes apparent. Namely, these rules will be our main tools in reducing the sizes of parts of $G - X$ located between the important bags.

Twin reduction

Rule 6.3. *If $T \subseteq V(G)$ is a true twin class of size $|T| > 2k + 5$, and $v \in T$ is an arbitrarily picked vertex, then remove v from the graph, i.e., proceed with the instance $(G - v, k)$.*

Lemma 6.13. *Applicability of Rule 6.3 can be recognized in polynomial time. Moreover, Rule 6.3 is safe, i.e., (G, k) is a yes-instance if and only if $(G - v, k)$ is a yes-instance.*

Proof. In order to recognize the applicability of Rule 6.3 we only need to inspect every true twin classes in the graph, which clearly can be done in polynomial time. We proceed to the proof of the safeness of the rule.

Let T be a true twin class of size at least $2k + 5$ and let v be the vertex the rule deleted. Since the class of trivially perfect graphs is hereditary, if (G, k) is a yes-instance, it follows that $(G - v, k)$ is a yes-instance. Suppose now that $(G - v, k)$ is a yes-instance. Let F be a set of edges with $|F| \leq k$ such that $(G - v) \triangle F$ is trivially perfect. We now show that $G \triangle F$ is also trivially perfect, which means that F is also a solution to (G, k). For the sake of contradiction, suppose W is an obstruction in $G \triangle F$. Since $(G - v) \triangle F$ is trivially perfect, W must contain the deleted vertex v. Since F has size at most k, at most $2k$ vertices of T can be incident to an edge of F. Let v_1, v_2, v_3, and v_4 be four vertices of T that are different from v and are not incident to F. Then one of them, say v_1, is not contained in W. Since v and v_1 are true twins both in G and in $G \triangle F$, we can replace v with v_1 in W yielding a new set W' which is an obstruction in $G \triangle F$. However, since v is not a member of W', we have that W' is an obstruction in $(G - v) \triangle F$, contradicting the assumption that $(G - v) \triangle F$ was trivially perfect. □

Module reduction

Recall that a module is a set of vertices M such that for every vertex v in $V(G) \setminus M$, either $M \subseteq N(v)$ or $M \cap N(v) = \emptyset$; see Definition 2.8. The following rule enables us to reduce large trivially perfect modules.

Rule 6.4. *Suppose $M \subseteq V(G)$ is a module such that $G[M]$ is trivially perfect and it contains an independent set of size at least $2k + 5$. Then let us take any independent set $I \subseteq M$ of size $2k + 4$, and we delete every vertex of M apart from I, i.e., proceed with the instance $(G - (M \setminus I), k)$.*

Observe that Rule 6.4 always deletes at least one vertex, since $|M| \geq 2k + 5$ and $|I| = 2k + 4$. Actually, we could define a stronger rule where we only assume that $|M| \geq 2k + 5$; however, the current statement will be helpful in recognizing the applicability of Rule 6.4.

We first prove that the rule is indeed safe.

Lemma 6.14. *Provided that (G, k) is a reduced instance (w.r.t. Rules 6.1 and 6.2), then Rule 6.4 is safe, i.e., (G, k) is a yes-instance if and only if $(G - (M \setminus I), k)$ is a yes-instance.*

Proof. Let $A = M \setminus I$, and $G' = G - A$. Since G' is an induced subgraph of G, by heredity, if (G, k) is a yes-instance, then (G', k) is a yes-instance. We proceed to the proof of the other direction. Suppose then that (G', k) is a yes-instance, and let F, $|F| \leq k$, be a minimum-size editing set for G'.

Claim 6.15. *No vertex of I is incident to any edit of F.*

Proof of claim. Since F has minimum possible size, it is inclusion-wise minimal. We show that if $F_I \subseteq F$ is the set of edges of F incident to a vertex of I and $F' = F \setminus F_I$, then $G' \triangle F$ being trivially perfect implies $G' \triangle F'$ being trivially perfect. Since $|I| = 2k + 4$, we can find at least four vertices $v_1, \ldots, v_4 \in I$ that are not incident to any edit of F. Suppose that $G' \triangle F'$ is not trivially perfect. Then there is an obstruction W in $G' \triangle F'$ containing at least one of the vertices of I incident to an edge of F. Create W' by replacing every vertex of $(W \cap I) \setminus \{v_1, \ldots, v_4\}$ by a different vertex of $\{v_1, \ldots, v_4\}$ that is not contained in W. Since vertices of I are not incident to the edits of F', they are false twins in $G' \triangle F'$, and hence W' created in this manner induces a graph isomorphic to the one induced by W. Thus, W' is an obstacle in $G' \triangle F'$. However, the vertices $v_1, \ldots, v_4$ are not incident to the edits of F and hence W' induces the same graph in $G' \triangle F'$ as in $G' \triangle F$. Therefore W' would be an obstacle in $G' \triangle F$, a contradiction to $G' \triangle F$ being trivially perfect.

Since we argued that $F' \subseteq F$ is also a solution, by the optimality of F we infer that $F = F'$ and $F_I = \emptyset$. ⌟

We now argue that $G \triangle F$ is trivially perfect, which will imply that (G, k) is a yes-instance. For the sake of contradiction, suppose that there exists an obstacle W in $G \triangle F$; It follows that W shares at least one vertex with $M \setminus I$. From Claim 6.15 it follows that no edit of F is incident to any vertex of M, so in $G \triangle F$ we still have that M is a module.

If the obstruction W induces a P_4, then it is known that W is fully contained in the module M, or has at most one vertex in M [GHPP13, Observation 1]. Since $G[M] = (G \triangle F)[M]$ is trivially perfect, the latter is the case. But since M is a module in $G \triangle F$, then replacing the single vertex of $W \cap A$ with any vertex of I would yield an obstacle in $G' \triangle F$, a contradiction.

Consider then the case when W induces a C_4 in $G \triangle F$. We have that W is not entirely contained in M since $G[M] = (G \triangle F)[M]$ is C_4-free. Also, if W had three vertices in M, then the remaining vertex would need to be contained in $N_G(M)$, and hence would be adjacent in $G \triangle F$ to all the other three vertices of W, a contradiction to $(G \triangle F)[W]$ being a C_4. Therefore, at most two vertices of W can be in M.

Suppose exactly two vertices w_1 and w_3 of W are in M, and w_2 and w_4 are outside M. As M is a module both in G and in $G \triangle F$, we must have that $w_2, w_4 \in N_G(M)$ and hence the four-cycle induced by W in $G \triangle F$ must be $\langle w_1, w_2, w_3, w_4 \rangle$. Take any two vertices $w_1', w_3' \in I$ and obtain W' by replacing w_1 and w_3 with them. It follows that W' induces a C_4 in $G' \triangle F$, a contradiction.

Finally, consider the case when exactly one vertex of W, say w_1, is in M. Again, replacing w_1 with any vertex of I would yield an induced C_4 contained in $G' \triangle F$, a contradiction. Thus, we conclude that $G \triangle F$ is trivially perfect. □

Observe that in order to apply Rule 6.4, one needs to be given the module M. Given M, finding any independent set $I \subseteq M$ of size $2k+4$ can then be done easily as follows: We can find an independent set of maximum cardinality in M in polynomial time, since $G[M]$ is trivially perfect and the Independent Set problem is polynomial-time solvable on trivially perfect graphs (it boils down to picking one vertex from every leaf bag of the universal clique decomposition of the considered graph). Then we take any of its subsets of size $2k+4$ to be I. Hence, to apply Rule 6.4 exhaustively, we need the following statement.

Lemma 6.16. *There exists a polynomial-time algorithm that, given an instance (G, k), either finds a module $M \subseteq V(G)$ where Rule 6.4 can be applied, or correctly concludes that Rule 6.4 is inapplicable.*

Proof. Using Proposition 2.9, we can compute the module decomposition of G, namely $(T, (M^t)_{t \in V(T)})$. Then we verify the applicability of Rule 6.4 to each module M^t for $t \in V(T)$, by checking whether $G[M]$ is trivially perfect and contains an independent set of size $2k+5$ (the latter check can be done in polynomial time since $G[M]$ is trivially perfect). Moreover, we perform the same check on all the modules N_t formed as follows: take a union node $t \in V(T)$, and construct a module N_t by taking the union of labels of those children of t that induce trivially perfect graphs.

We now argue that if Rule 6.4 is applicable to some module M in G, then this algorithm will encounter some (possibly different) module M' to which Rule 6.4 is applicable as well. By the third point of Proposition2.9, either $M = M^t$ for some $t \in V(T)$, or M is the union of a collection of labels of children of some union or join node. In the first case the algorithm verifies M explicitly. In the following, let $\alpha(H)$ denote the size of a maximum independent set in a graph H.

If now M is a union of labels of some children of a union node t, then by heredity $M \subseteq N^t$. Moreover, N^t induces a trivially perfect graph (since trivially perfect graphs are closed under taking disjoint union) and clearly $\alpha(N^t) \geq \alpha(M)$. Hence, Rule 6.4 is applicable to $M' = N^t$, and this will be discovered by the algorithm.

Finally, suppose M is a union of labels of some children $t_1, t_2, \ldots, t_p$ of a join node t. Observe that since for every $i \neq j$, every vertex of M^{t_i} is adjacent to every vertex of M^{t_j}, it follows that $\alpha(G[M]) = \max_{i=1,2,\ldots,p} \alpha(G[M^{t_i}])$. Without loss of generality suppose that the maximum on the right hand side is attained for the module M^{t_1}. Then by heredity $G[M^{t_1}]$ is trivially perfect, and $\alpha(G[M^{t_1}]) = \alpha(G[M]) \geq 2k+5$. Therefore Rule 6.4 is applicable to $M' = M^{t_1}$, and this will be discovered by the algorithm. □

We remark here that for the kernelization algorithm it is not necessary to be sure that Rule 6.4 is inapplicable at all. Instead, we could perform it on demand. More

precisely, during further analysis of the structure of $G - X$ we argue that some modules have to be small, since otherwise Rule 6.4 would be applicable. This analysis can be performed by a polynomial-time algorithm that would just apply Rule 6.4 on any encountered module that needs shrinking. However, we feel that the fact that Rule 6.4 can be indeed applied exhaustively provides a better insight into the algorithm, and streamlines the presentation.

Having introduced and verified Rules 6.3 and 6.4, we can now prove that after applying them exhaustively, all the trivially perfect modules in the graph are small.

Lemma 6.17. *A (possibly disconnected) trivially perfect graph with maximum true twin class size t and maximum independent set size α has at most $(2\alpha - 1)t$ vertices in total.*

Proof. Let $\mathcal{T}$ be the UCD of G, a trivially perfect graph with independent set number α and every true twin class of size at most t. Since any collection comprising one vertex from each leaf bag of $\mathcal{T}$ forms an independent set, there are at most α leaf bags in $\mathcal{T}$. Thus the number of nodes of $\mathcal{T}$ in total is at most $2\alpha - 1$. Since every bag of the decomposition $T \subseteq V(G)$ is a true twin class, we conclude that there are at most $(2\alpha - 1)t$ vertices in G. □

Corollary 6.18. *Suppose an instance (G, k) is reduced, and moreover Rules 6.3 and 6.4 are not applicable to (G, k). Then for every module $M \subseteq V(G)$ such that $G[M]$ is trivially perfect, we have that $|M| = O(k^2)$.*

Proof. Suppose M is such a module. Observe that members of every true twin class in $G[M]$ are also true twins in G (since M is a module). Hence twin classes in $G[M]$ have size at most $2k + 4$, as otherwise Rule 6.3 would be applicable. Moreover, if $G[M]$ contained an independent set of size $2k+5$, then Rule 6.4 would be applicable. By Lemma 6.17, we infer that $|M| \leq (4k + 7)(2k + 4) = O(k^2)$. □

From now on we assume that in the considered instance (G, k) we have exhaustively applied Rules 6.1–6.4, using the algorithms of Lemmata 6.1, 6.13, and 6.16. Hence Corollary 6.18 can be used. Observe that to perform this step, we do not need to construct the small modulator X at all. However, we hope that the reader already sees that Rules 6.1–6.4 will be useful for shrinking too large parts of $G - X$ between the important bags.

6.1.6 Irrelevant vertex deletion

Recall that we have fixed a small modulator X with $|X| \leq 4k$ such that $G - X$ is a trivially perfect graph with universal clique decomposition $\mathcal{T}$. Moreover, Rules 6.1–6.4 are inapplicable to (G, k). By Lemma 6.7 we have that the number of X-neighborhoods is $O(k^4)$. By the marking procedure, we have marked a set I of $O(k)$ bags of $\mathcal{T}$ as important, in such a manner that every connected component of $\mathcal{T} - I$ is adjacent to at most two vertices of I, and is in fact of one of the three forms described at the end of Section 6.1.4.

Thus, the whole vertex set of $G - X$ can be partitioned into four sets:

V_I: vertices contained in bags from I;

V_0: vertices contained in bags of those components of $\mathcal{T} - I$ that are not adjacent to any bag from I;

V_1: vertices contained in bags of those components of $\mathcal{T} - I$ that are adjacent to exactly one bag from I;

V_2: vertices contained in bags of those components of $\mathcal{T} - I$ that are adjacent to exactly two bags from I.

We are going to establish an upper bound on the cardinality of each of these sets separately. Upper bounds for V_I, V_0, and V_1 follow already from the introduced reduction rules, but for V_2 we shall need a new reduction rule. The upper bounds on the cardinalities of V_I and V_0 are quite straightforward.

Lemma 6.19. $|V_I| \leq O(k^6)$.

Proof. Consider for some $a \in I$ the bag B_a. Note that B_a is a module in $G - X$. By Lemma 6.7 there are only $O(k^4)$ possible X-neighborhoods among vertices of $G - X$. Hence, vertices of B_a can be partitioned into $O(k^4)$ classes w.r.t. the neighborhoods in X. Each such class is a module in G that is also a clique, and hence it is a true twin class. Since the twin reduction rule (Rule 6.3) is not applicable, each true twin class has size at most $2k + 5$, which implies that $|B_a| \leq O(k^5)$. As $|I| = O(k)$, we conclude that $|V_I| \leq O(k^6)$. □

We remark that using a more precise analysis of the situation in one bag B_a for $a \in I$, one can see that the X-neighborhoods of elements of B_a are nested, so there is only at most $|X| + 1 \leq 4k + 1$ of them. By plugging in this argument in the proof of Lemma 6.19, we obtain a sharper upper bound of $O(k^3)$ instead of $O(k^6)$. However, the upper bounds on $|V_0|$ and $|V_1|$ are $O(k^6)$ and $O(k^7)$, respectively, so establishing a better bound here would have no influence on the overall asymptotic kernel size. Hence, we resorted to a simpler proof of a weaker upper bound.

Lemma 6.20. $|V_0| \leq O(k^6)$.

Proof. Observe that V_0 is the union of bags of these connected components of $G-X$, whose universal clique decompositions (being components of $\mathcal{T}$) do not contain any important bag. By the definition of important bags, each such connected component C is a module in G, and clearly its neighborhood is entirely contained in X. Recall that by Lemma 6.7 there are only $O(k^4)$ possible different X-neighborhoods among vertices of $G - X$. Thus, we can group the connected components of $G[V_0]$ according to their X-neighborhoods into $O(k^4)$ groups, and the union of vertex sets in each such group forms a module in G. Since Rule 6.4 is not applicable, by Corollary 6.18 we have that each of these modules has size $O(k^2)$. Thus we infer that $|V_0| \leq O(k^6)$. □

To bound the size of V_1 we need a few more definitions. Suppose that C is a component of $\mathcal{T} - I$ that is adjacent to exactly one important bag $a \in I$. By the construction of I, we have that C is a tree rooted in a child of a. We shall say that C is *attached below* a. The union of bags of all the components of $\mathcal{T} - I$ attached below a will be called the *tassel rooted at* a. Thus, V_1 can be partitioned into $O(k)$ tassels.

Lemma 6.21. *For every $a \in I$, the tassel rooted at a has size at most $O(k^6)$.*

Proof. Let $C_1, C_2, \ldots, C_r$ be the components of $\mathcal{T} - I$ rooted at the children of a, whose union of bags forms the tassel rooted at a. Recall that none of the C_is contains any important bag. Therefore, from Lemma 6.9 we infer that for any C_i and any $x \in X$, either all the vertices from the bags of C_i are adjacent to x, or none of them. Thus, the union of bags of each C_i forms a module in G: The vertices in this union have the same X-neighborhood, and moreover their neighborhoods in $G - X$ are formed by the vertices from the bags on the path from a to the root of a's connected component in $\mathcal{T}$. Similarly as in the proof of Lemma 6.20, by Lemma 6.7 there are only $O(k^4)$ possible X-neighborhoods, so we can partition the components C_i into $O(k^4)$ classes with respect to their neighborhoods in X. The union of bags in each such class forms a module in G; since Rule 6.4 is not applicable, by Corollary 6.18 we infer that its size is bounded by $O(k^2)$. Thus, the total number of vertices in all the components C_i is at most $O(k^6)$. □

As $|I| = O(k)$, Lemma 6.21 immediately implies the following.

Lemma 6.22. $|V_1| \leq O(k^7)$.

We are left with bounding the cardinality of V_2. Let us fix any component C of $\mathcal{T} - I$ which is adjacent in $\mathcal{T}$ to two nodes of I. From the construction of I, it follows that C has the following form:

- C contains a path $P = \langle a_1, a_2, \ldots, a_d \rangle$ such that in $\mathcal{T}$, node a_d is a child of an important node $b^{\uparrow}$, and a_1 has exactly one important child $b^{\downarrow}$.
- For every $i = 1, 2, \ldots, d$, C contains also all the subtrees of $\mathcal{T}$ rooted in children of a_i that are different from a_{i-1} (where $a_0 = b^{\downarrow}$).

Such a component C will be called a *comb* (see Figure 6.8). The path P is called the *shaft* of a comb; the union of the bags of the shaft will be denoted by Q. The union of the bags of the subtrees rooted in children of a_i, apart from a_{i-1}, will be called the *tooth at* i, and denoted by R_i. Note that the subgraph induced by a tooth is not necessarily connected; it is, however, always non-empty by the definition of the universal clique decomposition. We also denote $R = \bigcup_{i=1}^{d} R_i$. By somehow abusing the notation, we will also denote $B_i = B_{a_i}$ for $i = 1, 2, \ldots, d$. The number of teeth d is called the *length* of a comb.

Since the comb C does not contain any important vertices, from Lemma 6.9 and the construction of I we immediately infer the following observation about the X-neighborhoods of vertices of the shaft and the teeth.

Lemma 6.23. *There exists two sets Y, Z with $Z \subseteq Y \subseteq X$ such that $N_X(u) = Y$ for every $u \in Q$ and $N_X(v) = Z$ for every $v \in R$.*

In particular, Lemma 6.23 implies that every tooth of a comb is a module. Hence, since Rule 6.4 is not applicable, we infer that $|R_i| = O(k^2)$ for $i = 1, 2, \ldots, d$. Also, observe that each B_i is a twin class, so by inapplicability of Rule 6.3 we conclude that $|B_i| \leq 2k + 5$ for each $i = 1, 2, \ldots, d$.

Since $\mathcal{T}$ is a forest and $|I| = O(k)$, it follows that in $\mathcal{T} - I$ there are $O(k)$ combs. As we already observed, for each comb the sizes of individual teeth and bags on the shaft are bounded polynomially in k. Hence, the only thing that remains is to show how to reduce combs that are long. In order to do this, we need one more definition: a tooth R_i is called *simple* if $G[R_i]$ is edgeless, and it is called *complicated* otherwise. We can now state the final reduction rule.

Rule 6.5. *Suppose C is a comb of length at least $(4k+3)^2$, and adopt the introduced notation for the shaft and the teeth of C. Define an index β as follows:*

(i) If at least $4k + 3$ teeth R_i are complicated, then we let $\beta = d$.

(ii) Otherwise, there is a sequence of $4k+3$ consecutive teeth $R_i, R_{i+1}, \ldots, R_{i+4k+2}$ that are simple. Let β be the index of the last tooth of this sequence, i.e., $\beta = i + 4k + 2$.

Having defined β, remove the tooth R_β from the graph and do not modify the budget. That is, proceed with the instance $(G - R_\beta, k)$.

Lemma 6.24. *Rule 6.5 is safe.*

Proof. Since $G - R_\beta$ is an induced subgraph of G, then we trivially have that the existence of a solution for (G, k) implies the existence of a solution for $(G - R_\beta, k)$. Hence, we now prove the converse. Suppose that F is a solution to $(G - R_\beta, k)$, that is, a set of edits in $G - R_\beta$ such that $(G - R_\beta) \triangle F$ is trivially perfect and $|F| \leq k$.

We will say that a tooth R_i is *spoiled* if any vertex of $R_i \cup B_i$ is incident to an edit from F, and *clean* otherwise. The first goal is to find an index α such that

(a) $1 < \alpha < \beta$,

(b) the teeth $R_{\alpha-1}$ and R_α are clean, and

(c) if any of the teeth $R_{\alpha+1}, R_{\alpha+2}, \ldots, R_\beta$ is complicated, then R_α is complicated.

Suppose first that β was constructed according to case (i), i.e., there are at least $4k + 3$ complicated teeth in the comb, and hence $\beta = d$. Out of these teeth R_i, at most one can have index 1, at most one can have index d, at most $2k$ can be spoiled (since $|F| \leq k$) and at most $2k$ can have the preceding tooth R_{i-1} spoiled. This leaves at least one complicated tooth R_i such that $1 < i < d$ and both R_i and R_{i-1} are clean. Then we can take $\alpha = i$; thus, property (c) of α is satisfied since R_α is complicated.

Suppose then that β was constructed according to case (ii), i.e., the following teeth are all simple:

$$R_{\beta-(4k+2)}, R_{\beta-(4k+1)}, \ldots, R_{\beta-1}, R_\beta.$$

Similarly as before, out of these $4k+3$ teeth, one has index β, one has index $\beta-(4k+2)$, at most $2k$ can be spoiled, and at most $2k$ can have the preceding tooth spoiled. Hence, among them there is a tooth R_i such that $\beta-(4k+2) < i < \beta$ and both R_i and R_{i-1} are clean. Again, we take $\alpha = i$; thus, property (c) is satisfied since all the teeth $R_{\beta-(4k+2)}, R_{\beta-(4k+1)}, \ldots, R_{\beta-1}, R_\beta$ are simple.

With α defined, we are ready to complete the proof of Lemma 6.24. To that aim, define $L = \bigcup_{i=\alpha-1}^{\beta} B_i \cup R_i$. Construct F' from F by removing all the edits that are incident to any vertex of L; clearly $|F'| \leq |F| \leq k$. We claim that F' is a solution to the instance (G, k), that is, that $G \vartriangle F'$ is trivially perfect. For the sake of a contradiction, suppose that $A \subseteq V(G)$ is a vertex set of size 4 such that $G \vartriangle F'[A]$ is a P_4 or a C_4. Let $A_0 = A \cap L$ and $A_1 = A \setminus A_0$.

Claim 6.25. *$|A_0| = 1$ or $|A_0| = 2$.*

Proof of claim. Suppose first that $A_0 = \emptyset$, so $A \subseteq V(G) \setminus L \subseteq V(G - R_\beta)$. Since $F \cap [V(G) \setminus L]^2 = F' \cap [V(G) \setminus L]^2$ and $R_\beta \subseteq L$, we have that the induced subgraph $G \vartriangle F'[A]$ is equal to the induced subgraph $(G - R_\beta) \vartriangle F[A]$. However, the graph $(G - R_\beta) \vartriangle F$ is trivially perfect, so it cannot have an induced P_4 or C_4; a contradiction.

Suppose now that $|A_0| \geq 3$. Since $A_0 \subseteq L$ and no edit of F' is incident to any vertex of L, we infer that there is no edit of F' between vertices of A: only at most one vertex of A does not belong to A_0. Therefore $G[A] = G \vartriangle F'[A]$ and $G[A]$ is an induced C_4 or P_4 in the graph G. However, $A_0 \subseteq L \subseteq V(G) \setminus X$, so $|A \cap X| \leq 1$. Thus, $G[A]$ would be an obstacle in G that has at most one common vertex with modulator X, a contradiction with the definition of a modulator (Definition 6.2). ⌟

To obtain a contradiction, we shall construct a set A'_0 satisfying the following properties:

(i) $A'_0 \subseteq R_{\alpha-1} \cup B_{\alpha-1} \cup R_\alpha \cup B_\alpha$;

(ii) $|A'_0| = |A_0|$ and $G[A'_0]$ is edgeless if and only if $G[A_0]$ is edgeless;

(iii) $|A_0 \cap Q| = |A'_0 \cap Q|$ and hence $|A_0 \cap R| = |A'_0 \cap R|$.

Let us define $A' = A_1 \cup A'_0$. For now we postpone the exact construction

Claim 6.26. *If A'_0 satisfies properties (i), (ii), and (iii), then $G \vartriangle F'[A]$ is isomorphic to $G \vartriangle F'[A']$.*

Proof of claim. By property (iii) there exists a bijection η between A_0 and A'_0 that preserves belonging to Q or R between the argument and the image. Extend η to A by defining $\eta(u) = u$ for $u \in A_1$; we claim that η is an isomorphism between $G \vartriangle F'[A]$ and $G \vartriangle F'[A']$. To see this, observe that since $A_0, A'_0 \subseteq L$, then we have that no vertex of A_0 or A'_0 is incident to any edit of F'. Moreover, in G, all the vertices of $L \cap R$ have the same neighborhood in $V(G) \setminus L$, and the same holds also for the vertices of $L \cap Q$. As the neighborhoods of these vertices in G and in $G \vartriangle F'$ are exactly the same, we infer that each vertex $u \in A_0$ is adjacent in $G \vartriangle F'$ to the same vertices of A_1 as the vertex $\eta(u)$ is.

To conclude the proof, we need to prove that η restricted to A'_0 is also an isomorphism between $G \vartriangle F'[A_0]$ and $G \vartriangle F'[A'_0]$. Again, A_0 and A'_0 are not incident to any edit of F', so $G \vartriangle F'[A_0] = G[A_0]$ and $G \vartriangle F'[A'_0] = G[A'_0]$. By Claim 6.25 we have that $|A_0| = 1$ or $|A_0| = 2$, and we conclude by observing that a pair of simple graphs with at most two vertices are isomorphic if and only if both of them are edgeless or both of them contain an edge, and in both cases any bijection between the vertex sets is an isomorphism. ⌟

We now argue that the existence of a set A'_0 satisfying properties (i), (ii), and (iii) leads to a contradiction. Recall that the teeth $R_{\alpha-1}$ and R_α are clean, which means that no vertex of $R_{\alpha-1} \cup B_{\alpha-1} \cup R_\alpha \cup B_\alpha$ is incident to any edit from F. Moreover, as $\beta > \alpha$, we have that $A' \subseteq V(G - R_\beta)$. By the construction of F' and A' we infer that $G \vartriangle F'[A'] = (G - R_\beta) \vartriangle F[A']$. By Claim 6.26 we have that $G \vartriangle F'[A']$ is a P_4 or a C_4, since $G \vartriangle F'[A]$ was. This would, however, mean that $(G - R_\beta) \vartriangle F$ would contain an induced P_4 or an induced C_4, a contradiction to the assumption that $(G - R_\beta) \vartriangle F$ is trivially perfect.

Therefore, we are left with constructing a set A'_0 satisfying properties (i), (ii), and (iii). We give different constructions depending on the alignment of the vertices of A_0. In each case we just define A'_0; verifying properties (i), (ii), and (iii) in each case is trivial.

Case 1. $|A_0| = 1$.

- **Case 1a.** $A_0 = \{u\}$ and $u \in Q$. Then $A'_0 = \{u'\}$ for any $u' \in B_{\alpha-1}$.
- **Case 1b.** $A_0 = \{u\}$ and $u \in R$. Then $A'_0 = \{u'\}$ for any $u' \in R_{\alpha-1}$.

Case 2. $|A_0| = 2$.

- **Case 2a.** $A_0 = \{u, v\}$, $u, v \in Q$. As $G[Q]$ is a clique, it follows that $uv \in E(G)$. Then $A'_0 = \{u', v'\}$ for any $u' \in B_{\alpha-1}$ and $v' \in B_\alpha$.
- **Case 2b.** $A_0 = \{u, v\}$, $u \in Q$, $v \in R$, and $uv \notin E(G)$. Then $A'_0 = \{u', v'\}$ for any $u' \in B_{\alpha-1}$ and $v' \in R_\alpha$.
- **Case 2c.** $A_0 = \{u, v\}$, $u \in Q$, $v \in R$, and $uv \in E(G)$. Then $A'_0 = \{u', v'\}$ for any $u' \in B_\alpha$ and $v' \in R_{\alpha-1}$.
- **Case 2d.** $A_0 = \{u, v\}$, $u, v \in R$, and $uv \notin E(G)$. Then $A'_0 = \{u', v'\}$ for any $u' \in R_\alpha$ and $v' \in R_{\alpha-1}$.

Case 2e. $A_0 = \{u, v\}$, $u, v \in R$, and $uv \in E(G)$. As there are no edges in G between different teeth, we observe that $u, v \in R_i$ for some i such that $R_i \subseteq L$, i.e., $\alpha - 1 \leq i \leq \beta$. In particular, the tooth R_i must be complicated. If $i = \alpha - 1$ or $i = \alpha$, then we can take $A'_0 = A_0$. Otherwise we have that $\alpha < i \leq \beta$ and R_i is complicated, so by property (c) of β we infer that R_α is also complicated. Then we take $A'_0 = \{u', v'\}$ for any $u', v' \in R_\alpha$ such that $u'v' \in E(G)$.

This case study is exhaustive due to Claim 6.25. □

We can finally gather all the pieces and prove our main theorem.

Theorem 3. *The problem* Trivially Perfect Editing *admits a proper kernel with $O(k^7)$ vertices.*

Proof. The algorithm first applies Reduction Rules 6.1—6.4 exhaustively. As each application of a reduction rule either decreases n and does not change k, or decreases k while not changing n, the number of applications of these rules will be bounded by $O(n + k)$ until k becomes negative and we can conclude that we are working with a no-instance. By Lemmata 6.1, 6.13, 6.14, and 6.16, these rules are safe, applicability of each rule can be recognized in polynomial time, and applying the rules also takes polynomial time.

After Rules 6.1–6.4 have been applied exhaustively, we construct a small modulator X using the algorithm of Lemma 6.3. In case the construction fails, we conclude that we are working with a no-instance. Otherwise, in polynomial time we construct the universal clique decomposition $\mathcal{T}$ of $G - X$, and then we mark the set I of important bags. Both locating the important bags and performing the lowest common ancestor closure can be done in polynomial time. After this, we examine all the combs of $\mathcal{T} - I$. In case there is a comb of length greater than $(4k + 3)^2$, we apply Rule 6.5 on it and restart the whole algorithm. Observe that each application of this rule reduces the vertex count by one while keeping k, so the total number of times the algorithm is restarted is bounded by the vertex count of the original instance.

We are left with analyzing the situation when Reduction Rule 6.5 is not applicable, i.e., all the combs have length less than $(4k + 3)^2$. As we have argued, the inapplicability of Rules 6.3 and 6.4 ensures that bags of shafts of combs have sizes $O(k)$ and teeth of combs have sizes $O(k^2)$. Hence, every comb has $O(k^4)$ vertices. Since the number of combs is $O(k)$, we infer that $|V_2| \leq O(k^5)$. Together with the upper bounds on the sizes of V_I, V_0, and V_1 given by Lemmata 6.19, 6.20, and 6.22, we conclude that

$$|V(G)| = |X| + |V_I| + |V_0| + |V_1| + |V_2| \leq 4k + O(k^6) + O(k^6) + O(k^7) + O(k^5) = O(k^7).$$

Hence, we can output the current instance as the obtained kernel. □

6.2 The remaining problems

We now present how the technique applied to Trivially Perfect Editing also yields polynomial kernels for Trivially Perfect Completion and Trivially Perfect Deletion after minor modifications. That is, we prove the following theorem:

Theorem 4. Trivially Perfect Completion *and* Trivially Perfect Deletion *admit proper vertex kernels on* $O(k^7)$ *vertices.*

We show that all the rules given above, with only two minor modifications are correct for both problems. Clearly, the running times of the algorithms recognizing applicability of the rule do not depend on the problem we are solving, so we only need to argue for their safeness.

In the first two rules, Rules 6.1 and 6.2, we add and delete an edge, respectively, and the argument is that any editing set of size at most k must necessarily include this edit. However, in the completion and deletion version, we are not allowed both operations. Hence, for the first rule, in the deletion variant we can immediately infer that we are working with a no-instance, and respectively for the second rule in the completion variant.

Thus, the two following rules replace Rule 6.1 for deletion and Rule 6.2 for completion, and their safeness is guaranteed by a trivial modification of the proof of Lemma 6.1:

Rule 6.1D. *For an instance* (G, k) *with* $uv \notin E(G)$*, if there is a matching of size at least* $k+1$ *in* $\overline{G[N(u) \cap N(v)]}$*, then return a trivial no-instance as the computed kernel.*

Rule 6.2C. *For an instance* (G, k) *with* $uv \in E(G)$ *and* $N_1 = N(u) \setminus N[v]$ *and* $N_2 = N(v) \setminus N[u]$*, if there is a matching in* $\overline{G}$ *between* N_1 *and* N_2 *of size at least* $k+1$*, then return a trivial no-instance as the computed kernel.*

Observe that Rules 6.1D and 6.2C are applicable in exactly the same instances as their unmodified variants. Hence, exhaustive application of the basic rules with any of these modifications results in exactly the same notion of a reduced instance as the one introduced in Section 6.1.1. We now argue that Rules 6.3 and 6.4 are safe for both the deletion and the completion variant, without any modifications.

Lemma 6.27. *Rules 6.3 and 6.4 are safe both for* Trivially Perfect Completion *and for* Trivially Perfect Deletion.

Proof. The proof of the safeness of Rule 6.3 (Lemma 6.13) in fact argues that every editing set F for $(G - v, k)$ with $|F| \leq k$ is also an editing set for (G, k). This holds also for editing sets that consist only of edge additions/deletions, so the reasoning remains the same for Trivially Perfect Completion and Trivially Perfect Deletion.

The proof of the safeness of Rule 6.4 (Lemma 6.14) first argues that any minimum-size editing set F for the reduced instance (G', k) is not incident to any

vertex of I. This is done by showing that otherwise F would not be an inclusion-wise minimal editing set (proof of Claim 6.15), and the argumentation can be in the same manner applied to minimum-size completion/deletion sets. Then it is argued that F is in fact an editing set for the original instance (G, k), and the argumentation is oblivious to whether F is allowed to contain edge additions or deletions. □

We now proceed to the analysis of Rule 6.5 in the completion and deletion variants. First, let us consider the construction of the modulator. In the completion/deletion variants we can construct the modulator in exactly the same manner as for editing. Indeed, the main argument for the bound $|X| \leq 4k$ states that if the construction was performed for more than k rounds, then we are dealing with a no-instance, since then any editing set for G has size at least $k + 1$. Completion and deletion sets are editing sets in particular, so the same argument holds also for Trivially Perfect Completion and Trivially Perfect Deletion.

Results of Sections 6.1.3 and 6.1.4, i.e., the analysis of the X-neighborhoods and marking of the important bags, work in exactly the same manner, since they are based on the same notions of a reduced instance and of a modulator. Thus, Lemma 6.7 holds as well, and we have marked the same set I of $O(k)$ important bags, with the same properties. Rules 6.3 and 6.4 are not modified, so the bounds on $|V_I|$, $|V_0|$ and $|V_1|$ from Lemmata 6.19, 6.20, and 6.22 also hold.

We are left with analyzing Rule 6.5, and we claim that this rule is also safe for Trivially Perfect Completion and Trivially Perfect Deletion without any modifications. Indeed, in the proof of the safeness of the rule (Lemma 6.24), we have argued that for every editing set F ($|F| \leq k$) for the new instance (G', k), there exists some $F' \subseteq F$ which is a solution to the original instance (G, k). In case F consists of edge deletions or edge additions only, so does F'. Hence, (G', k) being a yes-instance of Trivially Perfect Completion, resp. Trivially Perfect Deletion, implies that (G, k) is also a yes-instance of the same problem. Thus Rule 6.5 is safe without any modifications, and the kernel size analysis contained in the proof of Theorem 3 (end of Section 6.1.6) can be performed in exactly the same manner. This concludes the proof of Theorem 4.

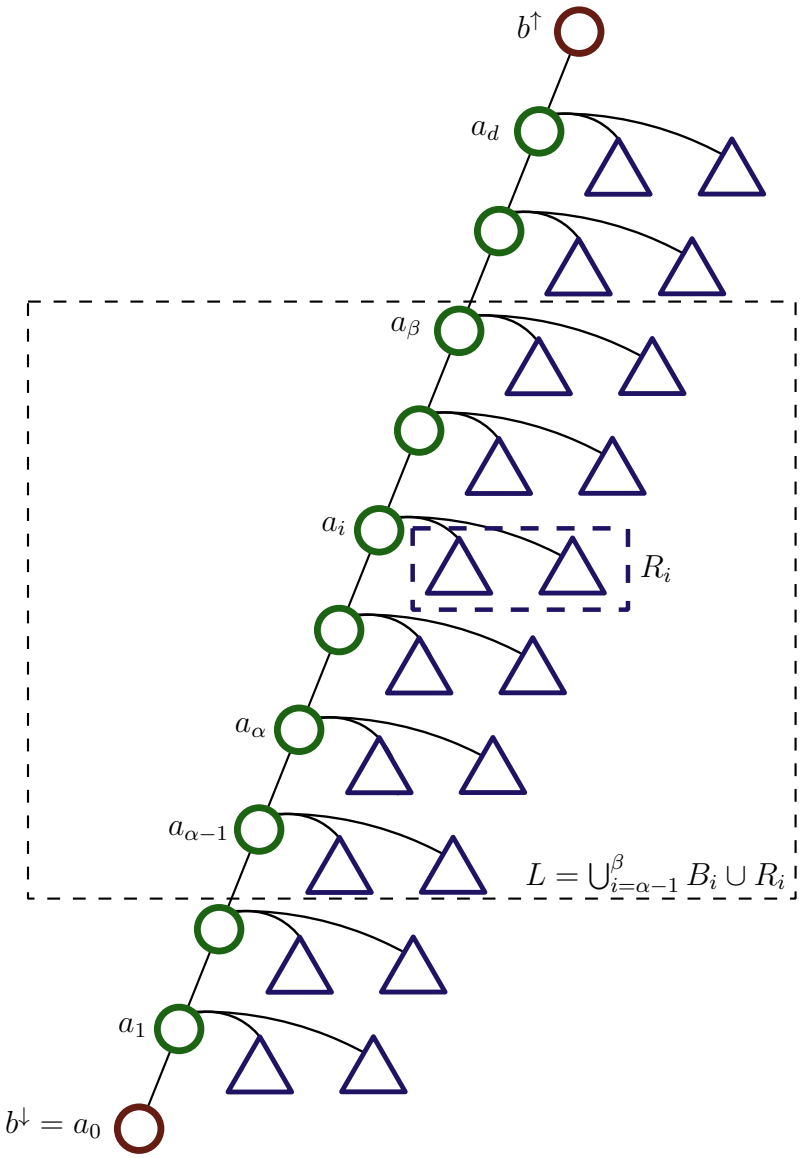


Figure 6.8: The anatomy of a *comb*. The top and bottom bags, $b^{\uparrow}$ and $b^{\downarrow}$, are important bags.

Chapter 7

Bicluster and related problems

A bicluster graph is a graph whose connected components are complete bipartite graphs, also called bicliques. We can lift this definition to t-partite cluster graphs, in which every connected component is a complete t-partite graph. Bicluster graphs have wide applications, especially in the context of bipartite similarity graphs and gene expression data, and the editing problem towards bicluster graphs, Bicluster Editing has been studied in great depth [CC99, Ami04, MO04, TSS05, GHKZ08].

We show a simple $O(ktp)$ kernel for the t-partite p-cluster Editing problem that will be the foundation of the subsequent subexponential parameterized time algorithms in Chapter 12. The kernelization algorithm has one single rule, Rule 7.1, which can be exhaustively applied in time $O(n+m)$. The problem at hand is the following generalization of p-Bicluster Editing:

t-partite p-cluster Editing parameterized by p, k

Input:	A graph $G = (V, E)$ and a non-negative integer k.
Question:	Is there a set F of at most k edges such that $G \triangle F$ is a disjoint union of exactly p complete t-partite graphs?

For our rule, we say that a set $T \subseteq V(G)$ is a *non-isolate twin class* if for every v and v' in T, $N_G(v) = N_G(v') \neq \emptyset$. Note that this is by definition a *false twin class*, i.e., $vv' \notin E(G)$, or in other words, a non-isolate twin class is an independent set.

Rule 7.1. *If there is a non-isolate twin class $T \subseteq V(G)$ of size at least $2k+2$, then delete all but $2k+1$ of the vertices of T. That is, let $T' \subseteq T$ be an arbitrary set such that $|T| = 2k+1$. Continue with the instance $(G - T', k)$.*

Lemma 7.1. *Rule 7.1 is sound and can be exhaustively applied in linear time.*

Proof. To reduce the number of connected components by one we need to add at least one edge. Hence, a yes-instance cannot contain more than $p+k$ connected components.

It is sufficient to observe that a non-isolated class of false twins T of size at least $2k+1$ will never be touched by a minimal solution. Let (G, k) be a yes-instance with F a solution. Suppose T is a non-isolated class of false twins of

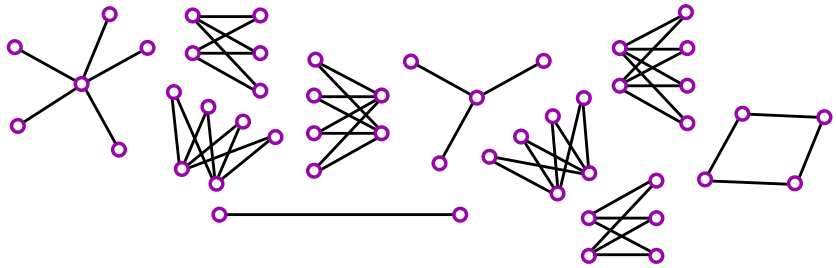


Figure 7.1: A bicluster

size at least $2k+1$. At most $2k$ vertices are touched by T, and we claim that F', the set of edges of F not incident to any vertex of T, is a solution. Let $x \in T$ be a vertex not incident with F. This means that $N_G(x)$ is exactly the entire complete t-partite component, with the exception of the vertices of the side to which x belongs. But since t-partite p-clusters are closed under adding non-isolated false twins, we may add as many false twins to x in G as we want without changing the solution. It follows that we may assume that its false twins will not be touched by F and hence F' is a solution as well.

The rule can be applied in linear time by first computing a modular decomposition of the input graph, which can be done in linear time [HP10], and marking all the vertices to be deleted. □

The following result is an immediate consequence of the above rules and their correctness.

Theorem 5. *The problem* t-PARTITE p-CLUSTER EDITING *admits a kernel with* $pt(2k+1)+2k = O(ptk)$ *vertices.*

Proof. We now count the number of vertices we can have in a yes-instance after the rules above have been applied. We claim that if G has more than $pt(2k+1)+2k$ vertices, it is a no-instance.

Let (G,k) be the reduced instance according to Rule 7.1 and let F be a solution of size at most k. At most $2k$ vertices can be touched by F, so the rest of the graph remains as it is, and is a disjoint union of at most p complete t-partite graphs, each of which has at most t non-isolate twin classes. It follows that in a yes-instance, G has at most $pt(2k+1)+2k = O(ptk)$ vertices. □

Chapter 8

On bounded degree input graphs

We know that there are $\mathcal{H}$-free Deletion and $\mathcal{H}$-free Editing problems that do not admit polynomial kernels unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ [KW13, GHPP13]. A different way of dealing with NP-completeness is as mentioned above, to restrict the input graph class, for instance, to graphs of bounded degree. However, there are problems that remain NP-hard on graphs on bounded degree, like Cluster Editing, Trivially Perfect Editing, Starforest Editing, etc.

These two limitations led Aravind, Sandeep, and Sivadasan [ASS14] to investigate the polynomial kernelizability of the problem $\mathcal{H}$-free Deletion, where $\mathcal{H}$ is a finite set of forbidden induced subgraphs, on bounded degree input graphs. They showed that as long as every graph in $\mathcal{H}$ is connected, the problem $\mathcal{H}$-free Deletion admits a polynomial kernel.

We will see in this chapter that the also $\mathcal{H}$-free Editing admits polynomial kernels, and we remove the requirement for the set $\mathcal{H}$ to contain only connected graphs. In the second section of this chapter, Section 8.2 we see that for a carefully constructed set $\mathcal{H}$, the problem $\mathcal{H}$-free Completion does not admit a polynomial kernel unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$, even on input graphs of degree at most 6.

8.1 Compressing bounded degree input

In this section we prove that for any finite set of obstructions $\mathcal{H}$, deleting or editing at most k edges to make an input graph of bounded degree $\mathcal{H}$-free admits polynomial kernels. More precisely, both $\mathcal{H}$-free Editing and $\mathcal{H}$-free Deletion admit polynomial kernels on bounded degree graphs.

The argument consists of two parts. First, we identify a set of critical vertices in the input graph G, called the obstruction core Z. Based on this set we can decompose any set of modifications F in G. The decomposition leads to the construction of a set of vertices in the graph, called the extended obstruction core Z^+. The first crucial property of Z^+ is that if F modifies $G[Z^+]$ into an $\mathcal{H}$-free graph, then F also modifies G into an $\mathcal{H}$-free graph. In other words, whichever obstructions we want to remove in the input graph should be done within the extended obstruction core. The second crucial property is that the extended

obstruction core can be proved to live within a ball around the obstruction core, were the radius depends on how well the solution decomposes. This ball will in the end constitute the kernel.

In the second part of the argument we prove that every minimal solution decomposes well. Hence we can bound the size of the ball containing the extended obstruction core and obtain a kernel.

We point out that we have considered the editing variant of the problem where we are allowed to surpass the original maximum degree in the graph by adding edges. However, it is true that there is always a solution that at most doubles the maximum degree of the graph since if more edges are added one might as well remove all edges incident to the vertex. The validity of this is proved in Lemma 8.11. Furthermore, it can be argued that the studied version of the problem is the most general one. This is due to the fact that adding every supergraph of the star with $\Delta(G) + 1$ leaves to the obstruction set ensures that any solution respects the current maximum degree.

8.1.1 Cores and layers

Now we introduce the concepts of obstruction cores and extended obstruction cores. They are heavily based on the notion of shattered obstructions, which are the set of obstructions we get from $\mathcal{H}$ if we take every connected component as an obstruction. It follows immediately that every shattered obstruction is connected.

Recall from the preliminaries that the size of the largest graph in $\mathcal{H}$ we denote by $n_{\mathcal{H}} = \max\{|V(H)| \text{ for } H \in \mathcal{H}\}$.

Definition 8.1 (Shattered obstructions)**.** Given a set of obstructions $\mathcal{H}$ we define the *shattered obstructions*, denoted $\mathcal{H}^\star$, as follows:

$$\{C : C \text{ is a connected component of } H \text{ and } H \text{ is a graph in } \mathcal{H}\}.$$

Based on shattered obstructions we now define an obstruction core and explain how such a set of not too large size can be obtained.

Definition 8.2 (Obstruction core)**.** Let (G, k) be an instance of $\mathcal{H}$-free Editing ($\mathcal{H}$-free Deletion). We say that a set $Z \subseteq V(G)$ is an *obstruction core in* G if for every shattered obstruction H in G it holds that either:

(i) $V(H) \subseteq Z$ or

(ii) there is an H-packing in $G[Z]$ of size at least $(\Delta(G) + 1) \cdot n_{\mathcal{H}} + 2k + 1$.

Observation 8.3. *Given an instance (G, k) of either $\mathcal{H}$-free Editing or of $\mathcal{H}$-free Deletion, we can in time $O(|\mathcal{H}^\star| n^{n_{\mathcal{H}}+1})$ obtain an obstruction core Z in G of size at most*

$$|\mathcal{H}^\star|((\Delta(G) + 1) \cdot n_{\mathcal{H}} + 2k + 1).$$

Proof. Let Z be the empty set initially. Then for every shattered obstruction H we find a maximal H-packing $\mathcal{X} = X_1, \ldots, X_t$ and add the following set $\bigcup_{i=1}^{p} X_i$ to Z, where $p = \min(t, (\Delta(G)+1) \cdot n_{\mathcal{H}} + 2k + 1)$. The time complexity follows by Observation 2.6. □

The next definitions are the ones of layer decompositions and core extensions, arguably the most central definitions of the kernelization algorithm. They are both with respect to a fixed obstruction core Z and set of edges F. The solution is decomposed into several layers such that the first layer consists of the edges of F that are contained in Z. The second layer consists of the edges of F that are contained in scattered obstructions created when the modifications in Z was done, and so forth. The extended core is a set of vertices encapsulating all scattered obstructions either in $G[Z]$ or created in G when doing the modifications of the layers.

Layer decompositions and core extensions. Let (G, k) be an instance of $\mathcal{H}$-free Editing ($\mathcal{H}$-free Deletion), $F \subseteq [V(G)]^2$ and Z an obstruction core. We construct the *layer decomposition* $F_1, \ldots, F_\ell$ of F as follows: Let $G_1 = G$, $R_1 = F$ and $Z_1 = Z$. Then, inductively we construct the set $X = R_i \cap [Z_i]^2$. If X is empty we stop the process, otherwise we let $F_i = X$, $G_{i+1} = G_i \triangle F_i$ and $R_{i+1} = R_i \setminus F_i$. Furthermore, we let

$$W_{i+1} = \{v \in H \mid H \text{ is a shattered obstruction in } G_{i+1} \text{ with } [V(H)]^2 \cap F_i \neq \emptyset\}.$$

Based on this we let $Z_{i+1} = Z_i \cup W_{i+1}$. With the construction above in mind we introduce some notation and terminology:

Definition 8.4 (Intermediate graphs and the extended core)**.** We will refer to G_i as the *i'th intermediate graph*, R_i as the *i'th remainder*, Z_i as the *i'th core extension* and ℓ as the *solution depth* (all with respect to G, Z and F). Furthermore, we will refer to $G^+ = G_{\ell+1}$ as the *resulting graph* and $Z^+ = Z_{\ell+1}$ as the *extended core*.

The next lemma says that if there is an obstruction in some intermediate graph such that every connected component of the obstruction is either inside the corresponding core extension or not modified at all so far by the layers, then there is an isomorphic obstruction contained entirely within the core extension. The intuition is that any untouched connected component has a large packing in Z and hence it can be replaced by an isomorphic subgraph inside Z that both avoids the modifications and the neighborhood of the rest of the obstruction.

Lemma 8.5. *Let (G, k) be an instance of either $\mathcal{H}$-free Editing, or of $\mathcal{H}$-free Deletion, Z an obstruction core of G, and $F \subseteq [V(G)]^2$ with $|F| \leq k$ and $F_1, \ldots, F_\ell$ a layer decomposition of F. For an integer $j \in [1, \ell+1]$ let G_j be the intermediate graph and Z_j the core extension with respect to G, Z and F. Let H be an obstruction in G_j with connected components $H_1, \ldots, H_t$ such that every H_i satisfies either:*

(i) $V(H_i) \subseteq Z_j$ *or*

(ii) $H_i = G[V(H_i)]$.

Then there is an obstruction H' *in* G_j *isomorphic to* H *with* $V(H') \subseteq Z_j$ *and* $V(H') \setminus V(H) \subseteq Z$.

Proof. For convenience we denote neighborhoods in G_j by N_j. Let H' be the disjoint union of every H_i such that $V(H_i) \subseteq Z_j$ and $\mathcal{L}$ the list containing every H_i not added to H'. Let H_i be an element of $\mathcal{L}$. We will now prove that there is an H_i' in $G_j[Z_j \setminus N_j[H']]$ such that H_i and H_i' are isomorphic. Let $\mathcal{X}_i$ be the maximal H_i-packing obtained when constructing Z. Since $V(H_i)$ is not contained in Z_j (and hence Z) and H_i's edges are as in G it holds that $|\mathcal{X}_i| \geq (\Delta(G)+1)\cdot n_{\mathcal{H}}+2k+1$ by the definition of obstruction cores. This yields that $(\Delta(G)+1)\cdot n_{\mathcal{H}}+2k+1$ of the elements of the packing was added to Z. Furthermore, we observe that $|V(H')| \leq n_{\mathcal{H}}$ and hence that $|N_G(H')| \leq \Delta \cdot n_{\mathcal{H}}$. It follows immediately that $|N_j(H')| \leq \Delta \cdot n_{\mathcal{H}} + k$ and hence that $|N_j[H']| \leq (\Delta+1)\cdot n_{\mathcal{H}} + k$. By the previous arguments it follows that there is an H_i-packing in $G_j[Z \setminus N_j[H']]$ of size at least $k+1$. And hence, by the pigeon hole principle there is an H_i' isomorphic to H_i in $G_j[Z_j \setminus N_j[H']]$ such that $[V(H_i')]^2$ and F' are disjoint.

To complete the proof we do the following for every H_i in $\mathcal{L}$. We find an H_i' as described above, add H_i' to H' and remove H_i from $\mathcal{L}$. Since $H_1, \ldots, H_t$ are the connected components of H it follows that H and H' are isomorphic. Furthermore, $V(H')$ is clearly contained in Z_j and $V(H') \setminus V(H)$ in Z. □

This possibility of moving obstructions to the inside of core extensions immediately yields several very useful lemmata.

Lemma 8.6. *Let* (G, k) *be an instance of either* $\mathcal{H}$-free Editing*, or of* $\mathcal{H}$-free Deletion*,* Z *an obstruction core of* G*, and* $F \subseteq [V(G)]^2$*. Construct the layer decomposition* $F_1, \ldots, F_\ell$ *of* F *with respect to* Z*, let* $F' = \cup_{i=1}^{\ell} F_i$ *and let* Z^+ *be the extended core with respect to* Z *and* F*. It then holds that:*

$$(G \triangle F')[Z^+] \text{ is } \mathcal{H}\text{-free if and only if } G \triangle F' \text{ is } \mathcal{H}\text{-free.}$$

Proof. Recall that $G^+ = G \triangle F'$. It is trivial that if there is an obstruction H in $G^+[Z^+]$ then H is also an obstruction in G^+. For the other direction, let H be an obstruction in G^+ and $H_1, \ldots, H_t$ the connected components of H. Observe that by the definition of Z^+ it holds that every H_i satisfies either *(i)* or *(ii)* of Lemma 8.5 with $j = \ell + 1$. It follows that there is an obstruction H' in G^+ with $V(H') \subseteq Z^+$. Hence H' is an obstruction in $G^+[Z^+]$, which completes the argument. □

Lemma 8.7. *Let* (G, k) *be an instance of either* $\mathcal{H}$-free Editing*, or of* $\mathcal{H}$-free Deletion*,* Z *an obstruction core of* G*,* F *a minimal solution and* $F_1, \ldots, F_\ell$ *the layer decomposition of* F *with respect to* Z*. It then holds that* $F_1, \ldots, F_\ell$ *forms a partition of* F*.*

Proof. Let F_i and F_j be two layers with $i < j$. It follows immediately from the definition of layer decomposition that $F_j \subseteq R_j \subseteq R_i \setminus F_i$ and hence F_i and F_j are disjoint. For convenience we let $F' = \cup_{i \in [1,\ell]} F_i$. We now prove that $F' = F$. It follows from the definition of layer decomposition that $F' \subseteq F$. Assume for a contradiction that $F' \subsetneq F$. Consider the final graph $G^+ = G \triangle F'$. If G^+ is $\mathcal{H}$-free it follows that F is not a minimal solution, yielding a contradiction.

Hence, G^+ is not $\mathcal{H}$-free. It follows immediately from Lemma 8.6 that $G^+[Z^+]$ is also not $\mathcal{H}$-free. Furthermore, we know by the definition of layer decompositions that $G^+[Z^+] = (G \triangle F)[Z^+]$. And hence $G \triangle F$ is not $\mathcal{H}$-free, contradicting that F is a solution. □

We finish the section by its two most crucial lemmata. The first one gives the true power of an extended core, namely that if a set of edges is a solution for the graph induced on its extended core it also is a solution for the entire graph. The second lemma gives us a partial tool for encapsulating an extended core without knowing the solution beforehand. The next section is dedicated to turning this partial tool into something useful.

Lemma 8.8. *Let (G, k) be an instance of either $\mathcal{H}$-free Editing, or of $\mathcal{H}$-free Deletion, Z an obstruction core of G, $F \subseteq [V(G)]^2$ and Z^+ the extended core with respect to Z and F. If $F \subseteq [Z^+]^2$ then*

$$(G \triangle F)[Z^+] \text{ is } \mathcal{H}\text{-free if and only if } G \triangle F \text{ is } \mathcal{H}\text{-free.}$$

Proof. Since $F \subseteq [Z^+]^2$ it holds that $G^+ = G \triangle F$. It trivially holds that if G^+ is $\mathcal{H}$-free, then so is $G^+[Z^+]$. Let H be an obstruction in G^+ with connected components $H_1, \ldots, H_t$. Observe that if H_i contains an edge of F then $V(H_i) \subseteq Z^+$ due to the definition of Z^+ and the assumption that $F \subseteq [Z^+]^2$. Apply Lemma 8.5 with $j = \ell + 1$ to obtain an obstruction H' in Z^+. □

Recall from the preliminaries that we let the notation of diameter account for a finite set of graphs $\mathcal{H}$, denoted $\operatorname{diam}(\mathcal{H})$, being the maximum of $\operatorname{diam} G$ for $G \in \mathcal{H}$, and whenever G is disconnected, the diameter is the maximum diameter of its connected components.

Lemma 8.9. *Let (G, k) be an instance of either $\mathcal{H}$-free Editing, or of $\mathcal{H}$-free Deletion, Z an obstruction core of G, $F \subseteq [V(G)]^2$ and Z^+ the extended core with respect to Z and F. It then holds that*

$$Z^+ \subseteq B(Z, \ell \cdot \operatorname{diam}(\mathcal{H})).$$

Proof. Let $Z_1, \ldots, Z_{\ell+1}$ be the extended cores. Instead of proving the lemma directly we prove the following, stronger claim:

($\star$) For every Z_i it holds that $Z_i \subseteq B(Z, (i-1) \cdot \operatorname{diam}(\mathcal{H}))$.

Since $Z^+ = Z_{\ell+1}$, it is clear that $(\star)$ implies the lemma. The proof of $(\star)$ is by induction. First, we observe that $(\star)$ holds for $i = 1$ by the definition of balls, since $Z = Z_1$. Assume for the induction step that $(\star)$ holds for i. Let v be a vertex in Z_{i+1}. If v is also in Z_i we are done by assumption. Hence, we assume v to be a vertex in $Z_{i+1} \setminus Z_i$. Or in other words, v is in W_{i+1}. By definition there is a scattered obstruction H in G_{i+1} and an edge uw in F_i such that both u, v and w are in H. Observe that the distance between u and v is at most $\mathrm{diam}(H)$ and recall that u is in $Z_i \subseteq B(Z, (i-1) \cdot \mathrm{diam}(\mathcal{H}))$. It follows immediately that v is in $B(Z, i \cdot \mathrm{diam}(\mathcal{H}))$ and hence the proof is complete. □

8.1.2 Solutions are shallow

In this section we prove that the depth of any solution is bounded logarithmically by the size of the solution. This, combined with Lemma 8.9 gives that linearly in k many balls of logarithmic radius is sufficient to encapsulate an extended core. To motivate that we obtain a polynomial kernel, observe that a ball of logarithmic radius in a bounded degree ball is of polynomial size.

First, we prove that when considering any layer we can always find a set of vertices of the same size, which removal would result in a $\mathcal{H}$-free graph. Next we prove that as long as the graph is not very small, removing a set of vertices from the graph has the same effect as modifying the graph such that the set becomes a set of isolates.

Lemma 8.10. *Let (G, k) be an instance of either $\mathcal{H}$-free Editing or of $\mathcal{H}$-free Deletion, with Z an obstruction core of G. Suppose that F is a minimal solution of the instance and $F_1, \ldots, F_\ell$ is the layer partition of F with respect to Z. For every $i \in [1, \ell]$ there exist a set Y with $Y \leq |F_i|$ such that $G_i - Y$ is $\mathcal{H}$-free.*

Proof. We construct Y as follows: For every edge uv in F_i we add to Z the endpoint furthest away from Z. If it is a tie, we choose an arbitrary endpoint. Assume for a contradiction that $G_i - Y$ is not $\mathcal{H}$-free. Let H be an obstruction in $G_i - Y$ and $H_1, \ldots, H_t$ the connected components of H.

First, we consider the case when $i = 1$. We then apply a modification of the proof of Lemma 8.5. The idea is as follows: Let H' be the disjoint union of the components of H contained in Z and H_x a component not in Z. Then there is a H_x-packing of size $k + 1$ in Z avoiding the closed neighborhood of H'. We observe that Y intersects with at most k of the elements of the packing and hence we can find a subgraph H'_x in $G[Z]$ not intersecting with Y such that H_x and H'_x are isomorphic. Add H'_x to H' and continue with the next component not contained in Z. It follows immediately that H' is also an obstruction in G_2. By definition $G_2[Z] = G^+[Z]$ and hence H' is an obstruction in G^+. This contradicts F being a solution.

If $i \geq 2$ it holds that Y and Z are disjoint. This is true since if both endpoints of an edge are included in Z, the edge would be in F_1 and not F_i. It holds by the definition of Y that $[V(H)]^2 \cap F_i$ is empty. Furthermore, by the definition of layer decompositions it holds that if some H_x intersects with some F_j with $j < i$

then $V(H_x) \subseteq Z_{j+1} \subseteq Z_i$. Hence we can apply Lemma 8.5 to obtain an obstruction H' in G_i with $V(H') \subseteq Z_i$. Since $V(H) \subseteq V(G) \setminus Y$ and $V(H') \setminus V(H) \subseteq Z$ it follows that H' is an obstruction in $G_i \setminus Y$. It follows immediately that H' is also an obstruction in G_{i+1}. By definition $G_{i+1}[Z_i] = G^+[Z_i]$ and hence H' is an obstruction in G^+. This contradicts F being a solution and completes the proof. □

We now show that if we the graph is big compared to the budget, the bounded degree, and $\mathcal{H}$, deleting vertices and deleting edges are in some sense equivalent.

Lemma 8.11. *Let (G, k) be an instance of either $\mathcal{H}$-free Editing, or of $\mathcal{H}$-free Deletion, X a set of vertices in G and E_X the set of edges incident to vertices in X. It then holds that either*

(i) $|V(G)| < |X| + k + 2(\Delta(G) + 1)n_{\mathcal{H}}$ *or*

(ii) the instances $(G - X, k')$ and $(G - E_X, k')$ are equivalent for every k'.

Proof. We assume that *(i)* does not apply and prove that this implies *(ii)*. It is trivial that if $(G - E_X, k')$ is a yes-instance then $(G - X, k')$ is also a yes-instance. For the other direction, assume for a contradiction that $(G - X, k')$ is a yes-instance and that $(G - E_X, k')$ is a no-instance. Let F be a solution of $(G - X, k')$. For convenience we define $G_V = (G - X) \triangle F$ and $G_E = (G - E_X) \triangle F$. Let H an obstruction in G_E and B the set of vertices $V(H) - V(H)$. Observe that $G_V[B] = G_E[B]$ and that $|N_{G_E}(V(H))| \leq \Delta(G) \cdot n_{\mathcal{H}} + k$. It follows immediately that

$$\begin{aligned}
&|V(G_E) \setminus (X \cup N_{G_E}[V(H)])| \\
&\geq |V(G_E)| - |X| - |N_{G_E}[V(H)]| \\
&\geq |X| + k + 2(\Delta(G) + 1)n_{\mathcal{H}} - |X| - n_{\mathcal{H}} - \Delta(G) \cdot LH - k \\
&= 2(\Delta(G) + 1)n_{\mathcal{H}} - n_{\mathcal{H}} - \Delta(G) \cdot n_{\mathcal{H}} \\
&= (\Delta(G) + 1)n_{\mathcal{H}}.
\end{aligned}$$

It follows immediately that we can obtain an independent set I of size $X \cap V(H)$ that is contained entirely outside of both X and $N_{G_E}[V(H)]$. Let $H' = G_V[I \cup B]$ and observe that H' is isomorphic to H, contradicting G_V being $\mathcal{H}$-free. □

With the two previous lemmata in mind we present the main intuition of the shallowness of a solution. Intuitively, if for any level of a decomposed solution we do a factor $\Delta(G)$ more modifications in the future than we do in this particular level, we could instead remove a set of edges related to this layer and stop any further propagation. This ensures that in any optimal solution the size of the union of the remaining layers are bounded by a layer and the maximum degree of the graph.

Lemma 8.12. *Given an instance (G, k) of either $\mathcal{H}$-free Editing, or of $\mathcal{H}$-free Deletion, an obstruction core Z, an optimal solution F and its layer decomposition $F_1, \ldots, F_\ell$, it holds that either*

(i) $|V(G)| \leq k + 2(\Delta(G)+1) \cdot n_{\mathcal{H}}$ *or*

(ii) $\Delta(G) \cdot |F_i| \geq |R_{i+1}|$ *for every* $i \in [1, \ell]$.

Proof. We assume that *(i)* does not apply and hence that

$$|V(G)| > k + (\Delta(G)+2) \cdot n_{\mathcal{H}}.$$

Assume for a contradiction that there is an $i \in [1, \ell]$ such that *(ii)* does not hold. Specifically, i is so that $\Delta(G) \cdot |F_i| < |R_{i+1}|$. By Lemma 8.10 there is a set of vertices Y with $|Y| \leq |F_i|$ such that $G_i - Y$ is $\mathcal{H}$-free. It follows by Lemma 8.11 with $k' = 0$ that $G_i - E_X$ is also $\mathcal{H}$-free. Let $F' = (\cup_{j \in [1,i-1]} F_j) \cup E_X$ and observe that $G \vartriangle F'$ is $\mathcal{H}$-free. By the following calculations:

$$|F'| \leq |\cup_{j \in [1,i-1]} F_j| + |E_X| < |\cup_{j \in [1,i-1]} F_j| + |R_{i+1}| = |F|$$

we conclude that $|F'| < |F|$. This contradicts the optimality of $|F|$ and hence our proof is complete. □

Lemma 8.13. *Given a instance* (G, k) *of either* $\mathcal{H}$-free Editing*, or of* $\mathcal{H}$-free Deletion*, an optimal solution* F *and its layer decomposition* $F_1, \ldots, F_\ell$*, it holds that either*

(i) $|V(G)| \leq k + 2(\Delta(G)+1) \cdot n_{\mathcal{H}}$ *or*

(ii) $\ell \leq 1 + \log_{\frac{\Delta(G)+1}{\Delta(G)}} |F|$.

Proof. Assume that *(i)* does not hold and hence that $|V(G)| > k + 2(\Delta(G)+1) \cdot n_{\mathcal{H}}$. It follows immediately that *(ii)* in Lemma 8.12 applies.

$$\begin{aligned}
|F| &= |R_1| = |F_1| + |R_2| \\
&\geq \frac{|R_2|}{\Delta(G)} + |R_2| = \frac{\Delta(G)+1}{\Delta(G)} \cdot |R_2| = \frac{\Delta(G)+1}{\Delta(G)} \cdot (|F_2| + |R_3|) \\
&\geq \cdots \geq \left(\frac{\Delta(G)+1}{\Delta(G)}\right)^{\ell-1} \cdot |R_\ell| \\
&= \left(\frac{\Delta(G)+1}{\Delta(G)}\right)^{\ell-1} \cdot |F_\ell| \\
&\geq \left(\frac{\Delta(G)+1}{\Delta(G)}\right)^{\ell-1}
\end{aligned}$$

This gives that $\ell \leq 1 + \log_{\frac{\Delta(G)+1}{\Delta(G)}} |F|$ and hence the argument is complete. □

8.1.3 Obtaining the kernels

We now have all the necessary tools for providing the kernels. We reduce the graph to a ball of small radius around any obstruction core Z and by this obtain a kernelized instance. Both the size bounds and the correctness of the reduction rule follows by combining the tools developed during the section.

Rule 8.1. *For a given instance (G, k) of either $\mathcal{H}$-free Editing, or of $\mathcal{H}$-free Deletion, and an obstruction core Z of G. If $|V(G)| > k + 2(\Delta(G)+1) \cdot n_{\mathcal{H}}$, return $(G[B(Z, r)], k)$ where $r = \operatorname{diam}(\mathcal{H}) \cdot (1 + \log_{\frac{\Delta(G)+1}{\Delta(G)}} k)$.*

The rule can clearly be applied in polynomial time; From Observation 8.3, we find Z in polynomial time and then it only remains to compute a breadth-first search from Z to depth r. This is the only rule, and thus the returned kernelized instance is exactly $(G[B(Z, r)], k)$, which yields a proper kernel.

We proceed now to prove that the rule is sound (Lemma 8.14) and that the kernel indeed is a polynomial kernel (Lemma 8.15) before we wrap up this section with Theorem 6.

Lemma 8.14. *Let (G, k) be an instance of either $\mathcal{H}$-free Editing, or of $\mathcal{H}$-free Deletion, and (G', k) the instance obtained when applying Rule 8.1 to (G, k). Then (G, k) is a yes-instance if and only if (G', k) is a yes-instance.*

Proof. It follows immediately from G' being an induced subgraph of G that if (G, k) is a yes-instance, then so is (G', k). For the other direction, let (G', k) be a yes-instance and let Z be the obstruction core of G obtained when applying Rule 8.1. Clearly, Z is also an obstruction core of G'. Let F be an optimal solution of (G', k) and construct the layer decomposition $F'_1, \ldots, F'_{\ell'}$ and the core extensions Z'_i with respect to Z and F in G'. Now we construct the layer decomposition $F_1, \ldots, F_\ell$ and the core extensions Z_i with respect to Z and F in G. By the definition core extensions it holds that $Z'_i \subseteq Z_i$ and hence $\ell \leq \ell'$. By Lemma 8.8 it holds that $Z_G^+ = Z_{\ell+1} \subseteq B_G(Z, \ell \cdot \operatorname{diam}(\mathcal{H})) \subseteq B_G(Z, \ell' \cdot \operatorname{diam}(\mathcal{H}))$. By Lemma 8.13 applied to F in G' it holds that

$$\ell \leq 1 + \log_{\frac{\Delta(G)+1}{\Delta(G)}} |F| \leq 1 + \log_{\frac{\Delta(G)+1}{\Delta(G)}} k,$$

and hence $Z_G^+ \subseteq V(G')$. It follows immediately that $(G \vartriangle F)[Z_G^+]$ is $\mathcal{H}$-free. By Lemma 8.7 it holds that $F \subseteq [Z'_{\ell'+1}]^2$ and hence $F \subseteq [Z_{\ell+1}]^2$. It follows immediately that Lemma 8.8 applies and hence $G \vartriangle F$ is $\mathcal{H}$-free. Hence (G, k) is a yes-instance and the proof is complete. □

For ease of readability, for the remainder of this section we denote $\operatorname{diam}(\mathcal{H})$ simply by D. The following lemma shows that the kernel given by the rule, is actually a polynomial kernel.

Lemma 8.15. *Let (G, k) be an instance of either $\mathcal{H}$-free Editing, or of $\mathcal{H}$-free Deletion, and (G', k) the instance obtained when applying Rule 8.1 to (G, k). Then the number of vertices in G' is at most*

$$2n_{\mathcal{H}}|\mathcal{H}^\star|\Delta^{D+1}k^{1+D/(\log_\Delta(\Delta+1)-1)}.$$

Proof. Clearly $|B(Z,r)| \leq |Z| \cdot \Delta^r$. By definition it holds that

$$|Z| \leq n_{\mathcal{H}}|\mathcal{H}^\star|((\Delta+1)+2k+1) \leq 2n_{\mathcal{H}}|\mathcal{H}^\star|\Delta k$$

for any non-trivial $\mathcal{H}$ and positive parameter k. To bound Δ^r, we observe that

$$\begin{aligned}\Delta^r &= \Delta^{D(1+\log_{\frac{\Delta+1}{\Delta}} k)} = \Delta^D \Delta^{D\cdot\log_{\frac{\Delta+1}{\Delta}} k} \\ &= \Delta^D \Delta^{D\cdot\log_\Delta k/\log_\Delta \frac{\Delta+1}{\Delta}} = \Delta^D k^{D/(\log_\Delta(\Delta+1)-1)},\end{aligned}$$

and hence

$$\begin{aligned}|V(G)| &\leq 2n_{\mathcal{H}}|\mathcal{H}^\star|\Delta k \cdot \Delta^D k^{D/(\log_\Delta(\Delta+1)-1)} \\ &= 2n_{\mathcal{H}}|\mathcal{H}^\star|\Delta^{D+1}k^{1+D/(\log_\Delta(\Delta+1)-1)}.\end{aligned}$$

□

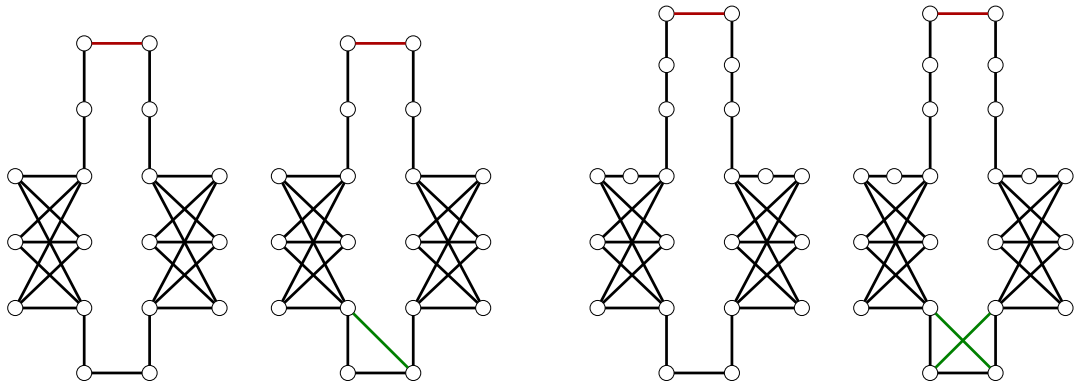
Theorem 6. *Both* $\mathcal{H}$-free Editing *and* $\mathcal{H}$-free Deletion *admit kernels with at most* $2n_{\mathcal{H}}|\mathcal{H}^\star|\Delta^{D+1}k^{1+D/(\log_\Delta(\Delta+1)-1)}$ *vertices. For fixed* $\mathcal{H}$ *and* Δ *this is a kernel where the number of vertices is bounded by* $\mathrm{poly}(k)$.

Proof. Given an instance (G,k) we find an obstruction core Z and apply Rule 8.1. By Observation 2.6 we can find Z in polynomial time. And by a standard breadth-first search we can apply Rule 8.1, given Z, in polynomial time. The correctness follows from Lemma 8.14 and the size of the kernel by Lemma 8.15. □

8.2 Hardness for completion

This section is devoted to giving strong evidence that for the *completion operation*, there cannot be a general polynomial kernelization result on bounded degree graphs, even when the target graph class is characterized by a finite sets of forbidden connected graphs. This strong evidence will come from the *cross-composition framework* in the form of an or-cross-composition (Definition 2.36). Proposition 2.37 stated that if an NP-complete language admits an or-cross-composition into a parameterized language, then the latter language does not have a polynomial kernel, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. To this aim, we give a finite set $\mathcal{H}$ of connected graphs for which $\mathcal{H}$-free Completion does not admit a polynomial kernel unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. The result here is purely a classification result, as it will be clear that the size of $\mathcal{H}$—albeit finite—is quite large. We will throughout this section refer to $\mathcal{H}$-free Completion with the intended meaning that $\mathcal{H}$ is a finite set of connected forbidden induced subgraphs determined later.

We use or-cross-composition and reduce from Cubic Planar Vertex Cover. In this problem we are given a planar cubic graph G (i.e., $\delta(G) = \Delta(G) = 3$), and an integer k' and asked to find a vertex cover of size k', that is, decide whether $\mathrm{vc}(G) \leq k'$. Observe that $\mathrm{vc}(G) \geq m/3$ since every vertex can cover at most three edges. Hence if $k' < m/3$, where m is the number of edges of G, we can safely reject the input. Since the input graph is cubic we may henceforth assume that $k' = \Theta(n+m)$.



(a) Selector tree forbidden (b) Selector tree ok (c) Duplicator forbidden (d) Duplicator ok

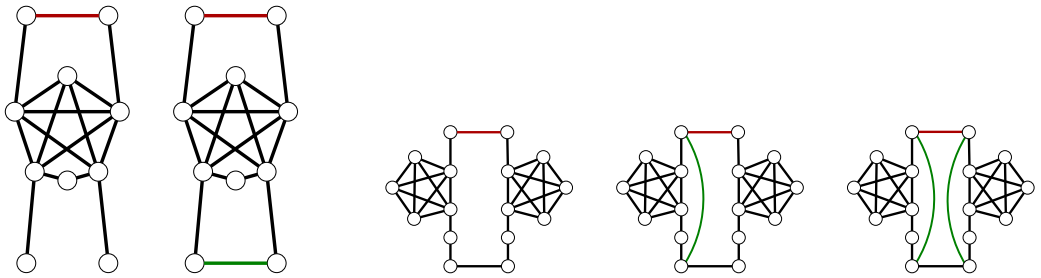
Figure 8.1: The selector tree gadget (on the left) and the duplicator gadget (on the right). The former has two possible completions, either to the left, or to the right. The latter (the duplicator) has only one legal completion, adding both edges.

Outline. We will define four *base graphs*. These graphs will be our main building blocks, and they are all non-planar, in the way that they contain, as a minor, a K_5 or a $K_{3,3}$. This means that we do not have to worry that they will appear in our problem instance to Cubic Planar Vertex Cover, whose graphs are all planar. The base graphs will all be added to $\mathcal{H}$ together with all their supergraphs, except for a few supergraphs. These supergraphs will act as selectors, or as propagators or duplicators when only one completion is allowed. The size of $\mathcal{H}$ will be bounded by the sum over all possible supergraphs for each of the base graphs. The base graphs are the following:

1. Selector tree (Figure 8.1a, one of the allowed completions is depicted in Figure 8.1b)
2. Duplicator gadget (Figure 8.1c, the unique allowed completion is depicted in Figure 8.1d)
3. Propagator gadget (Figure 8.2a, the unique allowed completion is depicted in Figure 8.2b)
4. Vertex selector gadget (Figure 8.2c, one of the allowed completions is depicted in Figures 8.2d and 8.2e)

Let $U_{\texttt{SEL}}$ be the selector tree gadget as depicted in Figure 8.1a. We define $U_{\texttt{SEL}}\uparrow$ to be the set of all supergraphs of $U_{\texttt{SEL}}$ *except* the two graphs isomorphic to the completed selector tree, depicted in Figure 8.1b. For the three other base graphs, we do the same thing. Let $U_{\texttt{DUP}}$ be the duplicator gadget and $U_{\texttt{DUP}}\uparrow$ all supergraphs except the one depicted in Figure 8.1d. Finally we construct $U_{\texttt{PRO}}\uparrow$ and $U_{\texttt{VER}}\uparrow$ for the propagator and vertex selector gadgets. These sets together comprise $\mathcal{H}$: Let

$$\mathcal{H} = U_{\texttt{SEL}}\uparrow \cup U_{\texttt{DUP}}\uparrow \cup U_{\texttt{PRO}}\uparrow \cup U_{\texttt{VER}}\uparrow .$$



(a) Propagator forbidden (b) Propagator ok (c) Vertex selector forbidden (d) Vertex selector ok (e) Vertex selector ok

Figure 8.2: The propagator gadget (on the left) and the vertex selector gadget (on the right). The propagator has as purpose to simply separate two gadgets. The vertex selector gadget is the final gadget used in the vertex cover reduction. Three completion of the latter gadget is allowed, shortcutting either the left or the right edge, or shortcutting both, corresponding to adding one or two endpoints of an edge to the vertex cover.

The OR-cross-composition will take as input $t = 2^r$ instances of the NP-complete problem Cubic Planar Vertex Cover (recall Proposition 2.14) where we may assume they are on the form $(G_1, k'), (G_2, k'), \ldots, (G_t, k')$, that is the parameter is the same across all instances, in addition to that $|V(G_i)| = |V(G_j)|$ for all $i, j \leq t$, hence also $|E(G_i)| = |E(G_j)|$. The restriction that t is a power of two is not necessary—we could attach any *cul-de-sac* gadget, e.g., the propagator with an extra edge in the K_5, on the end of the selector tree for every index $i > t$—but makes the proofs simpler to conceptualize.

We reduce to a single instance of $\mathcal{H}$-free Completion (G, k) with budget $k = \log t + k' + 3m - 2$. The graph G will have a single induced obstruction. The budget will be tight with $\log t$ edges forced in the selector gadget, selecting instance (G_i, k'), then $3m - 1$ edges will be used to construct the graph copy $\hat{G}_i$. The remaining part of the budget is the k' edges corresponding to a vertex cover of G_i. Recall that $m = \Theta(k')$ and hence $k = \text{poly}(k' + \log t)$.

8.2.1 Selector Tree

In this section we describe how to construct a selector tree that will be used in the OR-cross-composition. For now, let us fix t to be the number of instances provided as input to the reduction, and let $G_1, \ldots, G_t$ be the cubic planar graphs. Furthermore, denote by k' the budget for the input instance. We may assume that t is a power of two. Let $U_{\texttt{SEL}}$ be the graph depicted in Figure 8.1a. Denote by v_1 and v_2 the two top vertices of $U_{\texttt{SEL}}$. Denote the vertices on the bottom on the path (including endpoints) between the bicliques with a_1, b_2, a_2, b_1, in order.

Let T be a complete binary tree on $t/2 = 2^{r-1}$ leaves. Replace each leaf node ℓ_i with U_i. Note that this is one U_i for each two instances, i.e., we have

twice as many instances as we have leaves. For each two siblings U_1 and U_2 in the tree, replace its parent v by U_v and identify v_1, v_2 of U_1 with a_1 and a_2, and equivalently, identify v_1, v_2 of U_2 with b_1 and b_2. The tree construction T is depicted in Figure 8.3. Finally, for every U_i, except the one corresponding to the root, we remove the edge v_1v_2.

We define $h(T)$ to be the height of T, i.e., $h(T) = \log(t)$ (see Figure 8.3).

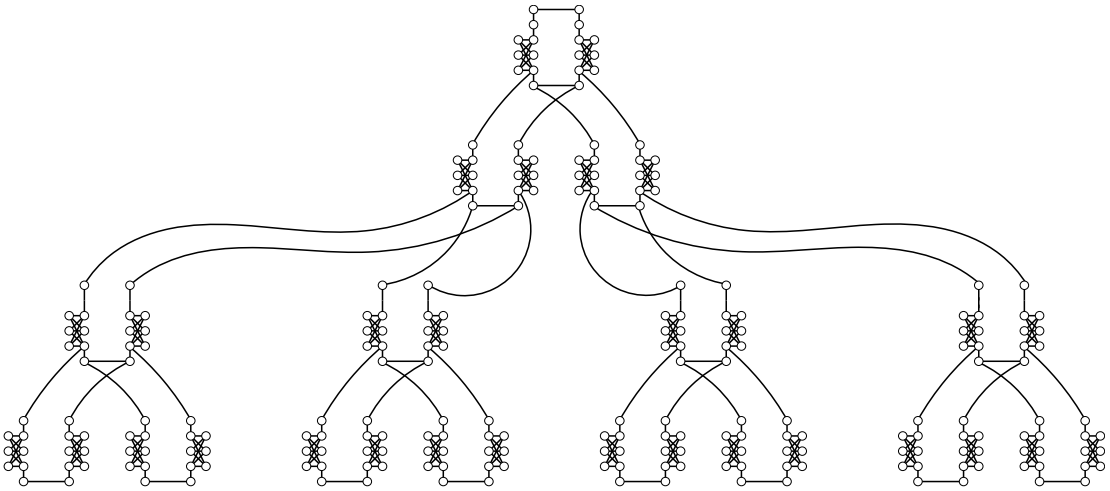


Figure 8.3: The selector tree T is a complete binary tree with nodes replaced by copies of $U_{\texttt{SEL}}$. For an input t (e.g. $t = 16$ in this case), we construct the complete binary tree with $t/2$ leaves. The tree has height $h(T) = \log(t) = 4$ and needs a budget of 4.

Lemma 8.16. *The constructed selector tree has maximum degree five.*

Proof. It is easy to verify that the vertices with maximum degree in T are the vertices corresponding to a_1 and b_1, i.e., the vertices of $K_{3,3}$ identified with top vertices in a child. These have degree 5. □

Definition 8.17. A solution F of (T, k) is said to *select* i if the $\lfloor i/2 \rfloor$th leaf has its corresponding a_1a_2 (or b_1b_2 if i is even) in F.

Lemma 8.18. *Given an instance $(T, h(T))$ of $\mathcal{H}$-free Completion, the following holds: $(T, h(T))$ is a yes-instance, and $(T, h(T) - 1)$ is a no-instance.*

Proof. By strong induction on $h(T)$. Let $h(T) = 1$, i.e., we have one copy of $U_{\texttt{SEL}}$. We know that $U_{\texttt{SEL}}$ is forbidden, and that U can be eliminated by adding one edge. This concludes the base case.

Suppose the statement is true for all $h(T') \leq n-1$. Construct T with $h(T) = n$ by taking two copies of T' of height $h(T) - 1$ and attaching them to a new root U_r. Clearly, again, we can take any solution of one of the subtrees and add a corresponding edge for U_r selecting the subtree with the solution. Suppose now that T had a solution of size $h(T) - 1 = h(T')$. We know that the root U_r must have one edge. But then $h(T')$ has a solution of size $h(T') - 1$, contradicting the induction hypothesis. See Figure 8.3. □

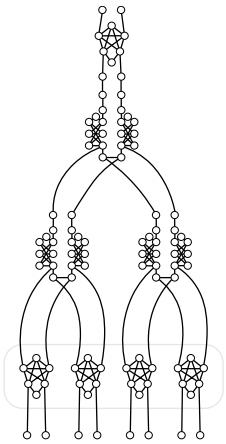


Figure 8.4: Instance activator M_i for $\hat{m} = 4$, with budget $b = 11 = 3\hat{m} - 1$.

With the tightness proven, we are ready to state the main lemma of the selector tree:

Lemma 8.19 (Selector lemma)**.** *Given an instance $(T, h(T))$ of $\mathcal{H}$-free* Completion*, any solution F of size at most $h(T)$ selects exactly one i.*

Proof. As witnessed in the proof above, every solution is on the form of a path from root to a leaf. For a budget of $h(T)$ edges, we can select exactly one leaf, which is the i in the statement. □

From Lemmata 8.18 and 8.19 we get the interface we wanted from this section; a tree T constructed as above, together with the corresponding budget, must be handled in one specific way, by adding one edge in all constructed $U_{\texttt{SEL}}$s in a path from a leaf to the root.

Corollary 8.20. *A budget of $\log(t) = h(T)$ is sufficient and necessary to eliminate all obstructions from T.*

8.2.2 Instance Activator

Let m be the number of edges in the instances we will reduce from and $\hat{m} < 2m$ the next power of two greater than m. In this section, we create for a given m, an instance activator M which consists of one propagator on the top, $\hat{m} - 1$ duplicators, and then $\hat{m}$ new propagators. The first propagator will be the interface to the *instance selector tree* in the previous section, Section 8.2.1, attached to an appropriate place in the selector tree.

We construct the graph M_i as seen in Figure 8.4, and then in the next section we construct P_i, for each instance G_i of Cubic Planar Vertex Cover.

Lemma 8.21. *When the ith propagator gadget is activated, all $\hat{m}$ edges out of M_i are activated and this uses budget $2(\hat{m} - 1) + 1 + \hat{m} = 3\hat{m} - 1$, not counting the activation edge on the top.*

Proof. The propagator gadget (Figure 8.2a) has only one possible completion (Figure 8.2b), and the same holds for the duplicator gadget (Figure 8.1c, the completion is depicted in Figure 8.1d). This implies that once the top propagator is activated, all edges in every duplicator will be added, and finally, again, all the $\hat{m}$ propagators will be activated. This sums up to the claimed budget. □

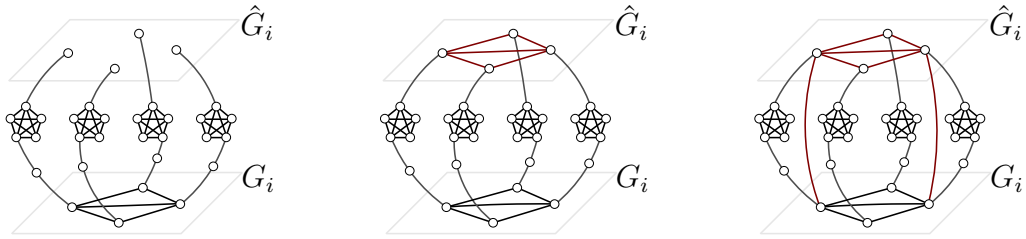
8.2.3 Cubic Planar Vertex Cover Reduction

With the instance activator M_i ready, we are ready to construct the actual vertex cover reduction. Let (G_i, k') be an instance of Cubic Planar Vertex Cover. In this section, we construct the output instance P_i of an input instance of Cubic Planar Vertex Cover.

We construct a copy $\hat{G}_i$ of G_i but with an empty edge set, hence, $\hat{G}_i$ is the empty graph on n vertices. Between $v \in V(G_i)$ and $\hat{v} \in V(\hat{G}_i)$, its corresponding vertex, we create a K_5, and connect v to a path of length 2 to the K_5 and $\hat{v}$ directly to a different vertex of the constructed clique. See Figure 8.5a.

Lemma 8.22. *When G_i is activated, P_i must be completed into $\hat{P}_i$ in any minimal completion.*

Proof. When the ith instance activator is activated, having a unique completion, must add every edge in $\hat{u}\hat{v}$ where the propagators are attached. □



(a) The planar graph gadget P_i before the edges of G_i has been added to $\hat{G}_i$.

(b) The completed graph, with edges added to $\hat{G}_i$. This depicts $\hat{P}_i$.

(c) The completed graph, with edges added to $\hat{G}_i$ and the vertex cover. This depicts $\hat{P}_i$ with vc.

Figure 8.5: The vertex cover gadget. On the left, P_i, the graph before the edges of $\hat{G}_i$ have been added. In the middle, the graph $\hat{P}_i$, when the edges of $\hat{G}_i$ have been added. On the right, the completed $\hat{P}_i$ with the vertex cover solution.

Lemma 8.23. *For (G_i, k) of* Cubic Planar Vertex Cover *the constructed graph P_i has $\Delta(P_i) \leq 5$.*

Proof. We attached one edge to each vertex of an empty graph and a cubic graph, so these instances have degree at most 4. The vertices in the K_5 attached to the rest of the gadgets have degree 5. These are the only vertices. □

Lemma 8.24. *P_i is $\mathcal{H}$-free.*

Proof. Since every forbidden structure contains as a minor either a K_5 or a $K_{3,3}$, we know that neither G_i nor $\hat{G}_i$ contains an obstruction. Hence the obstructions must use the newly introduced vertices.

We argue that none of the vertices in $\hat{G}_i$ are in any obstructions. There is only one forbidden structure with a pendant, namely the propagator. However, in P_i there are no propagators (only vertex selectors). Hence none of the vertices of G_i are in any forbidden structures and can thus be disregarded for the remainder of this proof. But $P_i - V(\hat{G}_i)$ is a planar graph with K_5s attached to each vertex which does not constitute any forbidden graphs. □

Let $\hat{P}_i$ be the graph obtained by adding every edge $\hat{v}\hat{u}$ for $uv \in E(G_i)$ to P_i.

Lemma 8.25. *$\hat{P}_i$ has exactly one obstruction per edge in G_i.*

Proof. Consider the possible obstructions that can live in $\hat{P}_i$. Since G_i and $\hat{G}_i$ are planar and there are no edges going between them, every obstruction needs to contain vertices between G_i and $\hat{G}_i$ (see Figure 8.5b). The graph $\hat{P}_i$ does not have any $K_{3,3}$s, nor $K_{3,3}$s with subdivided edges. Furthermore, it does not have any K_5s with a subdivided edge. Hence the only possible obstruction is the vertex selector (Figure 8.2c). Since the vertex selector is asymmetric—it has a path of length five on one side—we can observe that the obstructions must be on the form where vu is an edge and is the middle edge of the path of length five, and $\hat{v}\hat{u}$ must be the middle edge of the path of length three. There is one such obstruction per edge, and these are all the obstructions. □

Lemma 8.26. *$(\hat{P}_i, k')$ is a yes-instance of $\mathcal{H}$-free* Completion *if and only of G_i has a vertex cover of size k'.*

Proof. The first thing we want to observe is that for any obstruction using vu and $\hat{v}\hat{u}$, adding an edge $v\hat{v}$ or $u\hat{u}$ is sufficient to eliminate the obstruction, and furthermore does not create any new obstructions. The latter holds since any obstruction must be on the form vu and $\hat{v}\hat{u}$ and K_5s between v and $\hat{v}$, and u and $\hat{u}$. For the former statement, the following, even stronger statement holds: For any vertex $v \in G_i$, adding the edge $v\hat{v}$ will eliminate every obstruction in which v and $\hat{v}$ is.

With those two observations, we are ready to prove the lemma statement. In the forwards direction, let $(\hat{P}_i, k')$ be a yes-instance with F a solution. Since there is one obstruction per edge, and any solution will be of the form $\bigcup_{v \in V(F) \cap V(G_i)} \{v\hat{v}\}$, we will show that $\bigcup_{v \in V(F) \cap V(G_i)} \{v\}$ is a vertex cover of G_i. Suppose there is an edge uv which has not been covered by a vertex. Then uv and $\hat{u}\hat{v}$ together with the K_5s form an obstruction. This contradicts the assumption that F was a solution and concludes the forwards direction.

In the reverse direction, let C be a vertex cover of G_i of size at most k'. For each vertex $v \in C$, add the edge between v and $\hat{v}$, its corresponding vertex in $\hat{G}_i$. We claim that this graph is $\mathcal{H}$-free. Suppose there is still an obstruction. This obstruction has form uv and $\hat{u}\hat{v}$ with K_5s between. There are three allowed

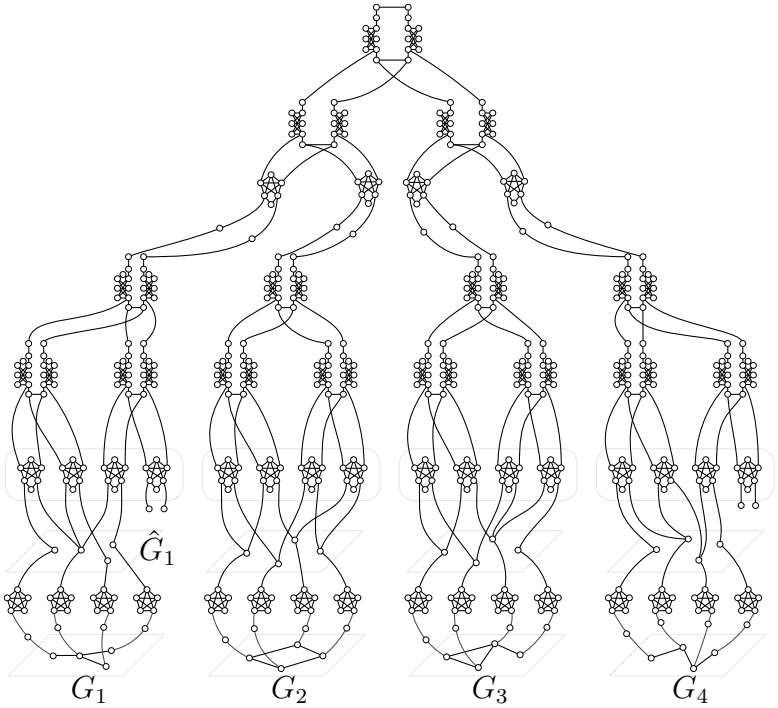


Figure 8.6: A complete reduction G for four graphs G_1, G_2, G_3, G_4. However, in these examples, the graphs are subcubic, and does not have the same number of edges. The intuition for the gadget is that if $k' = 1$ and G_1 has a vertex cover of size 1, then we can chose to complete the edges such that we solve G_1.

completions of the obstruction, namely of $\{u\hat{u}\}$, $\{v\hat{v}\}$, and $\{u\hat{u}, v\hat{v}\}$. However, none of them has been added by the construction, hence $u \notin C$ and $v \notin C$. It follows that C is not a vertex cover since uv is an edge.

This concludes the proof of the vertex cover reduction. □

8.2.4 Wrapping up the cross-composition

We now combine the gadgets from the previous sections, the selector tree from Section 8.2.1, the instance activator from Section 8.2.2 and the vertex cover reduction from the previous section, Section 8.2.3.

The goal is to have the tree T activate one instance activator M_i, which in turn adds all the edges of $\hat{G}_i$ in a vertex cover reduction P_i, thus constructing $\hat{P}_i$. This finally creates one induced copy of the forbidden *vertex selector gadget* (Figure 8.2c) for each edge of G_i. We here have a minimal completion corresponding directly to a vertex cover of G_i. This completion is depicted in Figure 8.2d.

Lemma 8.27. *Let G be the constructed instance above for input $(G_1, \ldots, G_t, k)$. The maximum degree of G is five.*

Proof. Again, it is not hard to find that the instance has degree bounded by 5 by inspecting all the gadgets and the way they are connected. Each vertex of T has

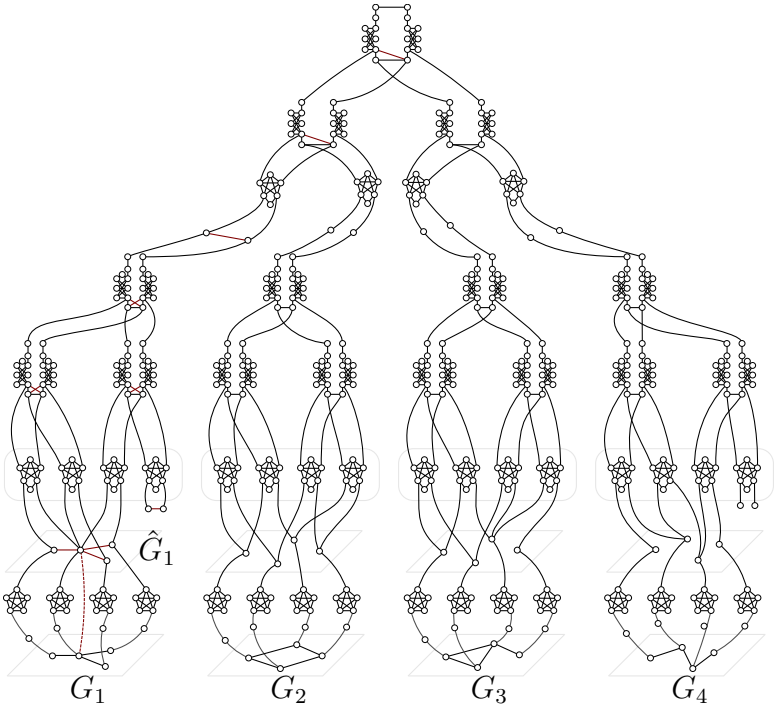


Figure 8.7: A completed reduction G where we have selected G_1 to be the instance we want to solve, and chosen the universal vertex of G_1 as our vertex cover. The budget is exactly $k' = \log t + 2m + k$.

degree at most 5 as per Lemma 8.16, the propagators have degree bounded by 5, the duplicators have degrees at most 5 (including the connections). Finally, P_i has degree bounded by 5, as the vertices in G_i all have degree at most 4 and the vertices in $\hat{G}_i$ have degrees at most 4. Finally, the connections in P_i between G_i and $\hat{G}_i$ have degrees 4 and 5. These are all the vertices of G. □

Lemma 8.28. *If (G_i, k) is a yes-instance, then F_T (the solution selecting i) together with a solution F_i for G_i and the four edges activating G_i is a solution of G.*

Proof. Invoke Lemmata 8.19, 8.26 and 8.27. □

Lemma 8.29. *Let (G, k, R) be a yes-instance constructed as above with $k = \log t + k' + 3m - 2$ and F a minimal solution of size at most k. Then there is an i such that:*

- *$F = F_i \cup F_T \cup$ the activation edges of G_i and*
- *(G_i, k) is a yes-instance of* Vertex Cover

Proof. Corollary 8.20 showed that it is necessary and sufficient with a budget of $\log t$ to eliminate all obstructions from T. Furthermore, Lemma 8.19 showed that there is exactly one i such that G_i is activated. When G_i has been activated, we need, by Lemmata 8.22 and 8.21 to add $3m - 1$ edges to P_i. Finally, we are

left with a completed $\hat{P}_i$, which by Lemma 8.26 is completable using at most k edges if and only if G_i is a yes-instance for Cubic Planar Vertex Cover. □

From the discussions in this section, it is clear that given a set of $t = 2^r$ many graphs $G_1, \ldots, G_t$ and a natural number k, we can construct in polynomial time an instance (G, k') which is a yes-instance for $\mathcal{H}$-free Completion if and only if there is an $i \leq t$ such that (G_i, k) is a yes-instance of Cubic Planar Vertex Cover. This is the "or" part of the definition of or-cross-composition (Definition 2.36). The second part of the definition is that the parameter value must be polynomially bounded ("pb"), and this is clear since $k' \leq 2k + \log t$. That is, (G, k) is subject to or-cross-composition.

As we observed in Lemma 8.27, $\Delta(G) = 5$, and hence applying Proposition 2.37 yields the following theorem.

Theorem 7. *Assuming* NP $\not\subseteq$ coNP/poly*, there exists a finite set* $\mathcal{H}$ *such that* $\mathcal{H}$-free Completion *does not admit a polynomial kernel, even on input graphs of maximum degree* 5*.*

Part III

Subexponential algorithms

In this part, we move away from the preprocessing aspect of parameterized complexity, and consider the actual solving of the problem. We focus on whether or not a parameterized problem is solvable in *subexponential parameterized time*, that is, in time $2^{o(k)} \cdot \text{poly}(n)$. The subexponential parameterized time complexity of graph modification problems is even more mysterious than the kernelization complexity and the classical complexity of edge modification problems. It seems, for some strange reason, that there are more natural completion problems than deletion problems that admit subexponential time algorithms, and sometimes it helps to bound the number of connected components in the solution [FKP+14, DRSVS15] and other times it does not [MPS13].

We will show that all the three modification problems are solvable in subexponential time for both threshold graphs and chain graphs. This puts these problems in a special category; These are to the best of the author's knowledge the only graph classes known for which editing, completion, and deletion are all NP-complete and subexponential time solvable. In the next part, Part IV, on lower bounds, we will show that neither Trivially Perfect Editing nor Deletion is solvable in subexponential time unless the exponential time hypothesis fails. However, the *completion version is* solvable in time $2^{O(\sqrt{k}\log k)} + \text{poly}(n)$, which we will see in Chapter 10.

Pseudosplit Editing is solvable in polynomial time as a consequence of Split Editing being solvable in polynomial time [HS81]. We will show this and that Pseudosplit Completion (and then also the deletion variant) are solvable in subexponential parameterized time.

Finally, we show that bounding the number of components in the solution by p gives an algorithm running in time $2^{O(\sqrt{pk})} + \text{poly}(n)$ for a "toy problem", Starforest Editing. In Chapter 16, Theorem 23 we show that the bound on the number of stars is essential in the quest for subexponential time algorithms. If we only parameterize by the budget, the problem does not have a subexponential time algorithm unless the exponential time hypothesis fails. The goal of that project was to obtain an algorithm matching that of p-Cluster Editing. However, we did not succeed in generalizing the algorithm for Starforest Editing to the "proper problem", Bicluster Editing. We managed only to achieve $2^{O(p\sqrt{k}\log pk)} + O(n+m)$ running time. It is an interesting open problem whether we can solve Bicluster Editing in time $2^{O(\sqrt{pk})} \cdot \text{poly}(n)$, as was done for Cluster Editing by Fomin et al. [FKP+14]. Note that the technique used by Fomin et al. is not immediately applicable for Bicluster Editing as the number of small cuts in a bicluster may be arbitrarily large.

Chapter 9

Threshold and chain graphs

In this chapter we show that there is an algorithm which in $2^{o(k)} + \text{poly}(n)$ solves the problems Threshold Editing, Threshold Completion, and by Fact 2.7, the equivalent problem Threshold Deletion. We then show, in Section 9.2, that we can modify the algorithm to work with Chain Editing as well as the corresponding deletion and completion problems. The chain graphs are not closed under pointwise complement. The complement of a chain graph is a *cobipartite chain graph*, but they are closed under *bipartite complement*. The bipartite complement of a bipartite graph $G = (A, B, E)$ is the graph $\overline{G} = (A, B, \overline{E})$, where $uv \in \overline{E}$ if and only if $|\{u, v\} \cap A| = 1$ and $uv \notin E$. In other words, we flip only edges between A and B.

9.1 Threshold graphs

In this section we will show that Threshold Editing is solvable in subexponential parameterized time. Later in the section we will see that a minor modification of our algorithm also solves both the completion and deletion variant as well. This section is devoted to the proof of the following theorem:

Theorem 8. Threshold Editing *admits a* $2^{O(\sqrt{k}\log k)} + \text{poly}(n)$ *time algorithm.*

The additive $\text{poly}(n)$ factor comes from the kernelization procedure of Chapter 4. The remainder of the algorithm operates on the kernel, and thus has running time that depends only on k.

We first refer to a *solution* F as a set such that, when given an instance (G, k), F is a set of at most k edges such that $G \triangle F$ is a threshold graph. In the next section, a solution will be such that $G \triangle F$ is a chain graph. After Section 9.1.1, we will be working with the problem Split Threshold Editing, so we assume $F \subseteq C \times I$ when (C, I) is the split partition of G.

SPLIT THRESHOLD EDITING parameterized by k

Input:	A split graph G with a split partition (C, I), and an integer k
Question:	Does there exist a set of at most k edges $F \subseteq [V]^2$ such that $G \triangle F$ is a threshold graph with split partition (C, I)?

Definition 9.1 (Potential split partition)**.** Given a graph G and an integer k, we call a partitioning (C, I) of $V(G)$ a *potential split partition* of G provided that

$$\binom{|C|}{2} - E(C) + E(I) \leq k.$$

That is, the cost of making G into a split graph with the prescribed partitioning does not exceed the budget.

A brief explanation of the algorithm for Theorem 8. The algorithm consists of four parts, the first of which is the kernelization algorithm described in Chapter 4. This gives in polynomial time an equivalent instance (G, k) with the guarantee that $|V(G)| = O(k^2)$. We may observe that this is a *proper kernel*, i.e., the reduced instance's parameter is bounded by the original parameter. This allows us to use time subexponential in the kernelized parameter.

The second step in the algorithm selects a potential split partitioning of G. We show that the number of such partitionings is bounded subexponentially in k, and that we can enumerate them all in subexponential time. This step actually also immediately implies that editing[1], completing and deleting to split graphs can be solved in subexponential time, however all of this was known [HS81, GKK+15]. The main part of this step is Lemma 9.2. For the remainder of the algorithm, we may thus assume that the input instance is a split graph, and that the split partition needs to be preserved, that is, we focus on solving SPLIT THRESHOLD EDITING.

The third and fourth steps of the algorithm consist of repeatedly finding special kind of separators and solving structured parts individually. Step three consists of locating *cheap vertices* (see Definition 2.16 for a formal explanation). These are the vertices, v, whose neighborhood is almost correct, in the sense that there is an optimal solution in which v is incident with only $O(\sqrt{k})$ edges. The dichotomy of cheap and expensive vertices gives us some tools for decomposing the graph. Specific configurations of cheap vertices allow us to extract three parts, one part is a highly structured part, the second part is a provably small part which we may brute force, and the last part we solve recursively. All of which is done in subexponential time $2^{O(\sqrt{k}\log k)}$.

Henceforth we will have in mind a "target graph" $H = G \triangle F$ with threshold partitioning $(\mathcal{C}, \mathcal{I})$ where F is the solution, i.e., $|F| \leq k$. See Definition 2.28 for the definition on threshold partitioning.

[1]Indeed, editing to split graphs is solvable in linear time [HS81].

9.1.1 Partitioning the graph

The second step of the subexponential time algorithm was as described above to compute the potential split partitionings of the input instance. Since we are given a general graph, we do not know immediately which vertices will go to the clique partition and which will go to the independent set partition. However, we now show that there is at most subexponentially many potential split partitionings. That is, there are subexponentially many partitionings of the vertex set into (C, I) such that it is possible to edit the input graph to a threshold graph with the given partitioning not exceeding the prescribed budget.

The next lemma will be crucial in our algorithm, as our algorithm presupposes a fixed split partition. Using this result, we may in subexponential time compute every possible split partition within range, and run our algorithm for editing to threshold graphs on each of these split graphs.

Lemma 9.2 (Few split partitions)**.** *There is an algorithm that, given a graph G and an integer k with $|V(G)| = k^{O(1)}$, can generate a set $\mathcal{P}$ of split partitions of $V(G)$ such that for every split graph H such that $|E(H) \triangle E(G)| \leq k$ and every split partition (C, I) of H it holds that (C, I) is an element of $\mathcal{P}$. Furthermore, the algorithm terminates in $2^{O(\sqrt{k}\log k)}$ time.*

Proof. Let $G = (V, E)$ be a graph and k a natural number. The first thing we do is to guess the size s_c of the clique and let C be a set of s_c vertices of highest degrees, and $s_i = n - s_c$, and let $I = V(G) \setminus C$. In the case that $\min\{s_c, s_i\} \leq 6\sqrt{k}$ we can simply enumerate every partitioning by Lemma 2.18, so we assume from now on that $\min\{s_c, s_i\} > 6\sqrt{k}$.

Claim 9.3. *Let H be a split graph with split partition (C', I') where $|E(H) \triangle E(G)| \leq k$. If $|C'| = s_c$, then $|C \triangle C'| \leq 2\sqrt{k}$ and $|I \triangle I'| \leq 2\sqrt{k}$.*

Proof of claim. Suppose that $2\sqrt{k}$ vertices $\hat{C} \subseteq C$ are in I' and that $2\sqrt{k}$ vertices $\hat{I} \subseteq I$ are in C'. Let $\sigma_c = \sum_{v \in \hat{C}} \deg(v)$ and $\sigma_i = \sum_{v \in \hat{I}} \deg(v)$. First, since the vertices in C of G are of degree at least that of the vertices in I of G, $\sigma_i \leq \sigma_c$. Second, since in the final solution, $\hat{C}$ is in the independent set, $\sigma_c \leq s_c 2\sqrt{k} + k$ (we can move at most k vertices from $\hat{C}$) and using the same reasoning, $\sigma_i \geq (s_c - 2\sqrt{k}) + \binom{2\sqrt{k}}{2} - k = s_c 2\sqrt{k} - 3k - \sqrt{k}$ (we can move at most k vertices to $\hat{I}$).

However, since $s_c > 6\sqrt{k}$, we have

$$s_c \cdot 2\sqrt{k} - 3k - \sqrt{k} \leq \sigma_i \leq \sigma_c \leq s_c \cdot 2\sqrt{k} + k, \text{ and thus}$$
$$9k - \sqrt{k} \leq \sigma_i \leq \sigma_c \leq 13k,$$

yielding that $\sigma_c \geq 9k - \sqrt{k}$. However, we can only lower the total degree of $\hat{C}$ by $2k$, which means that even if we spend the entire budget on deleting from $\hat{C}$, $\sum_{v \in C'} \deg_H(v) \geq 6k$ which means that there is a vertex in $\hat{C}$ with degree higher than the size of the clique (a contradiction). ⌟

Observe that since s_c and s_i are fixed, if we move ℓ vertices from C to I, we have to move ℓ vertices from I to C. Hence, if the claim holds, we can simply enumerate every set of $4\sqrt{k}$ vertices and take the sets with equally many on each side and swap their partition. Adding each such partition to $\mathcal{P}$ gives the set in question. □

We would like to remark that this lemma also gives a simpler algorithm for Split Completion (equivalently Split Deletion). Ghosh et al. [GKK+15] showed that Split Completion can be solved in time $2^{O(\sqrt{k}\log k)} \cdot \text{poly}(n)$ using the framework of Alon, Lokshtanov and Saurabh [ALS09]. However, the following observation immediately yields a very simple combinatorial argument for the existence of such an algorithm. Together with the polynomial kernel by Guo [Guo07], the following result is immediate from the above lemma.

Corollary 9.4. *The problem* Split Completion *is solvable in time* $2^{O(\sqrt{k}\log k)} + \text{poly}(n)$.

Proof. The algorithm is as follows. On (a kernelized [Guo07]) input (G, k) we compute, using Lemma 9.2, every potential split partitioning (C, I) at most k edges away from G. Then we in linear time check that I is indeed independent and that C lacks at most k edges from being complete. □

We will from now on assume that all our input graphs $G = (V, E)$ are split graphs provided with a split partition (C, I), and that we are to solve Split Threshold Editing, that is, we have to respect the split partitioning. We are allowed to do this with subexponential time overhead, as per Lemma 9.2. In addition, we assume that $|V(G)| = O(k^2)$.

9.1.2 Splitting Pairs and Unbreakable Segments

This section is devoted to showing that—in a yes-instance—either the graph at hand is small (compared to the parameter), or there is a splitting pair. We start by defining the splitting pairs and their counterparts, parts of the graph that do not contain splitting pairs. Finally we show how to greedily solve the latter, the unbreakable segments.

We start this section with three definitions. Recall from Section 2.1.3, and especially Definition 2.16, that given an instance (G, k) with solution F, a vertex v is called *cheap* if the edit number $\text{en}_F(v) \leq 2\sqrt{k}$.

Definition 9.5 (Splitting pair)**.** Let G be a graph, k an integer, F a solution of (G, k) and $(\mathcal{C}, \mathcal{I})$ a threshold decomposition of $G \triangle F$. We then say that the vertices $u \in I_a$ and $v \in C_b$ is a *splitting pair* if

- $a < b$,
- u and v are cheap,

- $\cup_{a<i<b} L_i$ consists of only expensive vertices. Recall from Definition 2.28 that $L_i = C_i \cup I_i$.

Definition 9.6 (Unbreakable)**.** Let G be a graph, k an integer, F a solution of (G, k) and $(\mathcal{C}, \mathcal{I})$ a threshold decomposition of $G \vartriangle F$. We then say that a sequence of levels $(C_a, I_a), (C_{a+1}, I_{a+1}), \ldots, (C_b, I_b)$ is an *unbreakable segment* if there is no splitting pair in the vertex set $\cup_{i\in[a,b]}(C_i \cup I_i)$.

Furthermore, we say that an instance (G, k) is *unbreakable* if there exists an optimal solution F and a threshold decomposition $(\mathcal{C}, \mathcal{I})$ of $G \vartriangle F$ such that the entire decomposition is an unbreakable segment. We also say that such a decomposition is a *witness* of G being unbreakable.

Definition 9.7 (Transfer level)**.** Let G be a graph and $(\mathcal{C}, \mathcal{I})$ a threshold decomposition of $G \vartriangle F$ for some solution F. Then we say that i is a *transfer level* if

- for every $j > i$ it holds that C_j contains no cheap vertices, and
- for every $j < i$ it holds that I_j contains no cheap vertices.

Given these three definitions, we are ready to investigate the structure of yes-instances of Split Threshold Editing. The remainder of this section is devoted to study how unbreakable segments and transfer levels relate, and bounding the number of levels in an unbreakable segment.

Lemma 9.8. *Let (G, k) be a yes-instance of* Split Threshold Editing *with solution F such that G is unbreakable and $(\mathcal{C}, \mathcal{I})$ is a witness. Then there is a transfer level in $(\mathcal{C}, \mathcal{I})$.*

Proof. Suppose for a contradiction that the lemma is false. Let a be the maximum index such that C_a contains a cheap vertex and b minimum such that I_b contains a cheap vertex. Since $i = a$ clearly satisfies the first condition, it must be the case that $b < a$. Increment b as long as $b+1 < a$ and there is a cheap vertex in $\cup_{i\in(b,a)} I_i$. Then decrement a as long as $b + 1 < a$ and there is a cheap vertex in $\cup_{i\in(b,a)} C_i$. Let u be a cheap vertex in C_a and v a cheap vertex in C_b. It follows from the procedure that they both exist. Observe that u, v is indeed a splitting pair, which is a contradiction to G being unbreakable and $(\mathcal{C}, \mathcal{I})$ being a witness. □

Lemma 9.9. *Let (G, k) be an instance of* Split Threshold Editing *such that G is unbreakable and $(\mathcal{C}, \mathcal{I})$ a witness of this. Then the number of levels in $(\mathcal{C}, \mathcal{I})$ is at most $2\sqrt{k} + 1$.*

Proof. Let i be the transfer level in $(\mathcal{C}, \mathcal{I})$. It is guaranteed to exist by Lemma 9.8. Observe that for every $j > i$ it holds that C_i consists of expensive vertices and for every $j < i$ it holds that I_i consists of expensive vertices. It follows immediately that every level besides i contains at least one expensive vertex. As there are at most $2\sqrt{k}$ such vertices the result follows immediately. □

We describe an algorithm, `unbreakAlg` which solves the unbreakable segments below. But first we need one final observation, namely stating how the cheap vertices appear in an unbreakable segment.

Lemma 9.10. *Let (G, k) be an instance of* Split Threshold Editing *such that G is unbreakable, $(\mathcal{C}, \mathcal{I})$ is a witness of this and F a corresponding solution. If X is the set of cheap vertices in G then $(G \triangle F)[X]$ forms a complete split graph.*

Proof. Let t be the transfer level of the decomposition, u a cheap vertex in C_i and v a cheap vertex in I_j for some i and j. By the definition of t it holds that $i \leq t \leq j$. It follows immediately that u and v are adjacent in $G \triangle F$ and the proof is complete. □

We will now describe the algorithm `unbreakAlg`. It takes as input an instance $(G, (C, I), k)$ of Split Threshold Editing, with the assumption that G is unbreakable and has split partition (C, I), and returns either an optimal solution F for (G, k) where $|F| \leq k$ or correctly concludes that (G, k) is a no-instance. Suppose that (G, k) is a yes-instance. Then there exists an optimal solution F and a threshold decomposition $(\mathcal{C}, \mathcal{I})$ of $G \triangle F$ that is a witness of G being unbreakable. First, we guess the number of levels ℓ in the decomposition, and by Lemma 9.9, we have that $\ell \in [0, 2\sqrt{k}+1]$ and the transfer level $t \in [0, \ell]$. Then we guess where the at most $2\sqrt{k}$ vertices that are expensive in G are positioned in $(\mathcal{C}, \mathcal{I})$. Observe that from this information we can obtain all edges between expensive vertices in F. Finally, we put every cheap vertex in the level that minimizes the cost of fixing its adjacencies into the expensive vertices while respecting that t is the transfer level. From this information we can obtain all adjacencies between cheap and expensive vertices in F. Since the set of cheap vertices induces a complete split graph, we reconstructed F and hence we return it.

Lemma 9.11. *Given an instance (G, k) of* Split Threshold Editing *with G being unbreakable,* `unbreakAlg` *either outputs an optimal solution or correctly concludes that (G, k) is a no-instance in time $2^{O(\sqrt{k}\log k)}$.*

Proof. Since the algorithm goes through every possible values for ℓ and t (according to Lemmata 9.8 and 9.9), and every possible placement of the expensive vertices, the only thing remaining to ensure is that the cheap vertices are placed correctly. However, since the cheap vertices form a complete split graph (according to Lemma 9.10), the only cost associated with a cheap vertex is the number of expensive vertices in the opposite side it is adjacent to. However, their placement is fixed, so we simply greedily minimize the cost of the vertex by putting it in a level that minimizes the number of necessary edits.

If we get a solution from the above procedure, this solution is optimal. On the other hand, if in every branch of the algorithm we are forced to edit more than k edges, then either (G, k) is a no-instance, or G is not unbreakable. Since the assumption of the algorithm is that G is unbreakable, we conclude that the algorithm is correct. □

9.1.3 Divide and Conquer

We now explain the main algorithm. The algorithm takes as input a graph G, together with a split partition (C, I) and a budget k. In addition, it takes a vertex set S which the algorithm is supposed to find an optimal solution for. The algorithm is recursive and either finds a splitting pair, in which it recurses on a subset of S, and if there is no splitting pair, then $G[S]$ is unbreakable, and thus it simply runs `unbreakAlg` on S. To avoid unnecessary recomputations, it uses memoization to solve already computed inputs.

The algorithm `solveAlg`$(G, (C, I), k, S)$ returns an optimal solution for the instance $(G[S], k)$, respecting the given split partition (C, I) in the manner described in Algorithm 1.

Algorithm 1 `solveAlg`$(G, (C, I), k, S)$

---

(1) Run `unbreakAlg`$(G[S], (C \cap S, I \cap S), k)$.

(2) For every pair of cheap vertices $u \in I$ and $v \in C$, together with their correct neighborhoods N_u and N_v, and every pair of subsets $C_X \subseteq C$ and $I_X \subseteq I$ of expensive vertices we do the following: Let $X = I_X \cup C_X$, $R_C = N_u$, $U_I = N_v \cap I$, $R_I = I \setminus (X \cup U_I)$ and $U_C = S \setminus (X \cup R_C \cup U_I \cup R_I)$. Now, $U = U_I \cup U_C$ is the unbreakable segment, X is the set of expensive vertices between the splitting pair, and $R = R_I \cup R_C$ is the remaining vertices. We now

- *(a)* Run `unbreakAlg`$(G[U], (C \cap U, I \cap U), k)$ yielding a solution F_U,
- *(b)* solve $G[X]$ optimally by brute force since it has size at most $2\sqrt{k}$, giving a solution F_X, and
- *(c)* recursively call `solveAlg`$(G, (C, I), k, R)$ to solve the instance corresponding to the remaining vertices yielding F_R.

Finally we return F, the union of F_U, F_X, and F_R together with all edges from $C \cap R$ and $I \cap (X \cup U)$, and all edges from $C \cap X$ to $I \cap U$.

In *(1)* we consider the option that there are no splitting pairs in G. In *(2)* (see Figure 9.1) we guess the uppermost splitting pair in the partition and the neighborhood of these two vertices. Then we guess all of the expensive vertices that live in between the two levels of the splitting pair. Observe that these expensive vertices together with the splitting pair partition the levels into three consecutive sequences. The upper one, U is an unbreakable segment, the middle, X are the expensive vertices and the lower one, R is simply the remaining graph.

When we apply `unbreakAlg` on the upper part, brute force the middle one and recurse with `solveAlg` on the lower part, we get individual optimal solutions for each of the three. Finally we may merge the solutions and add all the remaining edges (see end of *(2)*).

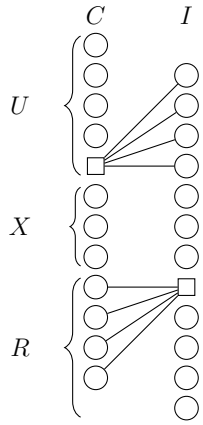


Figure 9.1: The partitioning of the vertex sets according to `solveAlg`. The square bags are the bags containing the splitting pair, U is an unbreakable segment and the bags of X contain exclusively expensive vertices. The edges drawn indicates the neighborhoods of the splitting pair across the partitions.

Lemma 9.12. *Given a split graph $G = (V, E)$ with split partition (C, I),* `solveAlg` *either returns an optimal solution for* Split Threshold Editing *on input $(G, (C, I), k, V)$, or correctly concludes that (G, k) is a no-instance.*

Proof. If (G, k) with split partition (C, I) is a yes-instance of Split Threshold Editing there is a solution F with threshold decomposition $(\mathcal{C}, \mathcal{I})$ and a sequence of pairs $(u_1, v_1), (u_2, v_2), \ldots, (u_t, v_t)$ such that u_1, v_1 is the splitting pair highest in $(\mathcal{C}, \mathcal{I})$, and u_2, v_2 in the highest splitting pair in the graph induced by the vertices in and below the level of v_1, etc. Since we in a state $(G, (C, I), k, S)$ try every possible pair of such cheap vertices and every possible neighborhood and set of expensive vertices, we exhaust all possibilities for any threshold editing of S of at most k edges. Hence, if there is a solution, an optimal solution is returned.

Thus, if ever an F is constructed of size $|F| > k$, we can safely conclude that there is no editing set $F^\star \subseteq C \times I$ of size at most k such that $G \triangle F^\star$ is a threshold graph. □

Lemma 9.13. *Given a split graph $G = (V, E)$ with split partition (C, I) and an integer k with $|V(G)| = O(k^2)$, the algorithm* `solveAlg` *terminates in time $2^{O(\sqrt{k} \log k)}$ on input $(G, (C, I), k, V)$.*

Proof. By charging a set S for which `solveAlg` is called with input $(G, (C, I), k, S)$ every operation except the recursive call, we need to *(i)* show that there are at most $2^{O(\sqrt{k} \log k)}$ many sets $S \subseteq V$ for which `solveAlg` is called, and *(ii)* that the work done inside one such call is at most $2^{O(\sqrt{k} \log k)}$.

For Case *(i)*, we simply note that when `solveAlg` is called with a set S, the sets R on which we recurse are uniquely defined by u, v, N_u, N_v, X, and there are at most $O(k^4) \cdot 2^{O(\sqrt{k} \log k)^3} = 2^{O(\sqrt{k} \log k)}$ such configurations, so at most $2^{O(\sqrt{k} \log k)}$ sets are charged. Case *(ii)* follows from the fact that we guess two vertices, u and v and three sets, N_u, N_v and X. For each choice we run `unbreakAlg`, which

runs in time $2^{O(\sqrt{k}\log k)}$ by Lemma 9.11, and the brute force solution takes time $2^{O(\sqrt{k}\log(\sqrt{k}))}$. The recursive call is charged to a smaller set, and merging the solutions into the final solution we return, F, takes polynomial time.

The two cases show that we charge at most $2^{O(\sqrt{k}\log k)}$ sets with $2^{O(\sqrt{k}\log k)}$ work, and hence `solveAlg` completes after $2^{O(\sqrt{k}\log k)}$ steps. □

To conclude, we observe that Theorem 8 follows directly from the above exposition. Given an input (G, k) to Threshold Editing, from the previous section we can in polynomial time obtain an equivalent instance with at most $O(k^2)$ vertices. Furthermore, by Lemma 9.2 we may in time $2^{O(\sqrt{k}\log k)}$ time assume we are solving the problem Split Threshold Editing. Finally, by Lemmata 9.12 and 9.13, the theorem follows.

Finally, since it is straight forward to observe that one can always choose F to be either disjoint from, or contained in the edge set of the input graph, $E(G)$, the following is an immediate consequence from the above and from the fact that both problems admit a polynomial kernel (Theorem 1):

Corollary 9.14. *The problems* Threshold Completion *and* Threshold Deletion *are solvable in time* $2^{O(\sqrt{k}\log k)} + \mathrm{poly}(n)$.

9.2 Chain graphs

We now proceed to give the necessary observations and constructions for obtaining subexponential algorithms for the edge modification problems to chain graphs using the results from the previous section. In other words, we show that Chain Editing, as well as the corresponding deletion and completion version, all can be solved in subexponential time.

The main difference between Chain Editing and Threshold Editing is that it is far from clear that the number of bipartitions is subexponential, that is, is there a bipartite equivalent of the bound of the potential split partitions as in Lemma 9.2? If we were able to enumerate all such "potential bipartitions" in subexponential time, we could simply run a very similar algorithm to the one above on the problem Bipartite Chain Editing, where we are asked to respect the bipartition (see Section 13.2 for the definition of this problem).

It turns out that we indeed are able to enumerate all such potential bipartitions within the allowed time:

Lemma 9.15. *There is an algorithm which, given a graph* $G = (V, E)$ *and an integer* k*, enumerates* $\binom{|V|}{O(\sqrt{k})} = 2^{O(\sqrt{k}\log|V|)}$ *bipartite graphs* $H = (A, B, E')$ *with* $|E \triangle E'| \leq k$ *with the following properties. If* (G, k) *is a yes-instance of* Chain Editing*, then at least one output* (H, k) *will be a yes-instance of* Bipartite Chain Editing*, and furthermore if any yes-instance* (H, k) *is output, then* (G, k) *is a yes-instance of* Chain Editing*. This also holds for the deletion and completion versions.*

Proof. We first mention that it is trivial to change the below proof into the proofs for the deletion and completion versions, one simply disallow one of the operations. So we will prove only the editing version. Furthermore, it is clear to see that if any output instance (H, k) is a yes-instance for Bipartite Chain Editing, then (G, k) was a yes-instance for Chain Editing.

Consider any solution $H = (A, B, E')$ for an input instance (G, k). If either A or B has size at most $5\sqrt{k}$, then we can simply guess every such in subexponential time. Hence, we assume that both sides of H are large. But this means, by Observation 2.17, that both A and B have cheap vertices. Let v_A be a cheap vertex as low as possible in A and v_B be a cheap vertex as high as possible in B. It immediately follows from the same observation that the set of vertices below v_A, A_X is a set of expensive vertices, and the same for the vertices above v_B, B_X. Since v_A and v_B are cheap, we can in subexponential time correctly guess their neighborhoods in H and we can similarly guess A_X and B_X.

Now, since we know v_A, v_B, $N_H(v_A)$ and $N_H(v_B)$, as well as A_X and B_X, the only vertices we do now know where to place, are the vertices in A which are in the levels above $\text{lev}(v_B)$, call them A_Y, and the vertices in b which are in the levels below $\text{lev}(v_A)$. However, we know which set this is, that is, we know $Z = A_Y \cup B_Y$. Define now $A_M = A \setminus (A_Y \cup A_X \cup \{v_A\})$ and similarly $B_M = B \setminus (B_Y \cup B_X \cup \{v_B\})$. These are the vertices living in the middle of A and B, respectively.

We now know that the vertices of Z should form an independent set. This follows from the fact that A_M and B_M are both non-empty. Hence, the vertices of A_Y are in higher levels than all of B_Y, and since there are no edges going from a vertex in A to a vertex lower in B, and each of A and B are independent sets, Z must be an independent set.

The following is the crucial last step. We can in subexponential time guess the partitioning of levels of both A_X and of B_X, since they are both of sizes at most $2\sqrt{k}$. When knowing these levels, we can greedily insert each vertex in Z into either A and B by pointwise minimizing the cost; A vertex $z \in Z$ can safely be places in the level of A or B which minimizes the cost of making it adjacent to only the vertices of B_X above its level, or by making it adjacent to only the vertices below its level in A_X. □

Given the above rather technical and rather cumbersome lemma, we may now work on the more restricted problem, Bipartite Chain Editing. It should be clear, judged by the similarities between Split Threshold Editing and Bipartite Chain Editing that the rest of the algorithm for Threshold Editing works without any changes.

Theorem 9. Chain Editing *is solvable in time* $2^{O(\sqrt{k}\log k)} + \text{poly}(n)$.

Proof. On input (G, k) we first run the kernelization algorithm from Chapter 5, and then we enumerate every potential bipartition according to Lemma 9.15. Now, for each bipartition (A, B) we make A into a clique, and run the Split Threshold Editing algorithm from the previous chapter.

Now, (G, k) is a yes-instance if and only if there is a bipartition (A, B) such that when making A into a clique, the resulting instance is a yes-instance for Split Threshold Editing. □

As for the completion and deletion version, they are both solvable in the same running time as the editing version, just as we observed in the previous section for threshold graphs. Together with the kernelization argument (Theorem 2), the above arguments immediately yield the following corollary:

Corollary 9.16. *The problem* Chain Completion*, and then also* Chain Deletion*, are solvable in subexponential parameterized time* $2^{O(\sqrt{k}\log k)} + \mathrm{poly}(n)$.

Chapter 10

Trivially perfect graphs

In this chapter we give an algorithm which, given an input instance (G, k) of Trivially Perfect Completion, in $2^{O(\sqrt{k}\log k)} + \text{poly}(n)$ time computes a completion set F of size at most k—a set such that $G + F$ is trivially perfect—or decides that (G, k) is a no-instance. The additive polynomial factor is purely the running time of the kernelization procedure. As mentioned in Chapter 6, a cubic vertex kernel was announced by Guo [Guo07]. Trivially Perfect Completion was shown to be NP-complete by Yannakakis [Yan81a].

Trivially Perfect Completion parameterized by k

Input:	A graph G and an integer k
Question:	Does there exist a set F of at most k edges such that $G + F$ is trivially perfect?

As already stated in the introduction and in the previous part on kernels, trivially perfect graphs are characterized by a finite set, $\{C_4, P_4\}$ of *forbidden induced subgraphs*, and thus it follows from Cai [Cai96] that the problem also is fixed parameter tractable, i.e., it belongs to the class FPT. The main result of this chapter is the following theorem:

Theorem 10. *For an input* (G, k)*,* Trivially Perfect Completion *is solvable in time* $2^{O(\sqrt{k}\log k)} + \text{poly}(n)$*.*

Throughout this chapter, an edge set F is called a *completion* for G if $G + F$ is trivially perfect. Furthermore, a completion F is called a *minimal completion* for G if no proper subset of F is a completion for G. The main outline of the algorithm is as follows:

Step A

On input (G, k), we first apply the algorithm by Guo [Guo07] to obtain a kernel $O(k^3)$ vertices. The running time of this algorithm is $O(kn^4)$. The kernelization algorithm of Guo can only reduce the parameter, i.e., $k' \leq k$ where k' is the new parameter. Moreover, the output kernel is in fact of size $O(k'^3)$. Therefore, due to this preprocessing step we may assume without loss of generality that we work on an instance (G, k) with $|V(G)| \leq O(k^3)$.

Step B

Assuming our input instance has $O(k^3)$ vertices, we show how to generate all special vertex subsets of the kernel which we call *vital potential maximal cliques* in time $2^{O(\sqrt{k}\log k)}$. A vital potential maximal clique $\Omega \subseteq V(G)$ is a vertex subset which is a maximal clique in some minimal completion of size at most k.

Step C

Using dynamic programming, we show how to compute an optimal solution or to conclude that (G, k) is a no-instance, in time polynomial in the number of vital potential maximal cliques.

10.1 Structure of minimal completions

In Section 2.2.3, we investigated the structure of trivially perfect graphs, and, in Definition 2.24 gave a decomposition of these graphs. This decomposition, the *universal clique decomposition*, or UCD, will be heavily used throughout this chapter. In this section, we will study the structure of yes-instances and properties of minimal completions.

Lemma 10.1. *Let $G = (V, E)$ be a connected graph, F a minimal completion of G, and let $H = G + F$. Suppose $L = (B, D)$ is a block in the universal clique decomposition of H and let $D_1, D_2, \ldots, D_\ell$ be the connected components of $H[D] - B$.*

(i) If L is not a leaf block, then $\ell > 1$;

(ii) If $\ell > 1$, then even in the original graph, G, every vertex $v \in B$ has at least one neighbor in each of the sets $D_1, D_2, \ldots, D_\ell$;

(iii) The graph $G[D_i]$ is connected for every $i \in \{1, \ldots, \ell\}$;

(iv) For every $i \in \{1, \ldots, \ell\}$, $B \subseteq N_G(D \setminus (B \cup D_i))$.

Proof. (i) Let (B, D) be a non-leaf block. Recall that by the definition of a universal clique decomposition, B is the maximal universal clique of $H[D]$. From Lemma 2.27 (i), we get that $B \neq D$. Since $B = \text{uni}(H[D])$, there must be two non-adjacent vertices in $D \setminus B$ and no universal vertex in $D \setminus B$. Since $H[D \setminus B]$ is trivially perfect, it must have several connected components, i.e., $\ell > 1$.

(ii) Suppose, without loss of generality, that there exists a vertex $v \in B$ such that $N_G(v) \cap D_1 = \emptyset$. Let $F' = F \setminus (\{v\} \times V(D_1))$. Note that since v is universal to $V(D_1)$ in H and is completely non-adjacent to $V(D_1)$ in G, it follows that $\{v\} \times V(D_1) \subseteq F$ and that F' is a proper subset of F. We claim that $H' = G + F'$ is also a trivially perfect graph, which contradicts the minimality of F. Indeed, consider a universal clique decomposition obtained from the universal clique decomposition of H by *(a)*, in case $\ell = 2$, moving v from B to the root bag of D_2,

or *(b)*, in case $\ell > 2$, moving v from B to a new bag $B' = \{v\}$ attached below B, with all the root bags of $D_2, D_3, \ldots, D_\ell$ re-attached from below B to below B'. It can easily be seen that this new universal clique decomposition is indeed a universal clique decomposition of H', which proves that H' is trivially perfect.

(iii) For the sake of a contradiction, suppose $G[D_a]$ was disconnected. Let (D_{a_1}, D_{a_2}) be a partition of D_a such that there is no edge between D_{a_1} and D_{a_2} in G. Clearly, $H[D_{a_1}]$ and $H[D_{a_2}]$ are trivially perfect graphs as they are induced subgraphs of H, and hence they admit some universal clique decompositions. Since $H[D_a]$ is connected, we infer that F contains some edges between D_{a_1} and D_{a_2}. Now let $F' = F \setminus \{uv \mid u \in D_{a_1}, v \in D_{a_2}, uv \in F\}$; by the previous argument we have that $F' \subsetneq F$. Modify now the given universal clique decomposition of H by removing the subtree below B that corresponds to D_a, and attaching instead two subtrees below B that are universal clique decompositions of $H[D_{a_1}]$ and $H[D_{a_2}]$. Observe that thus we obtain a universal clique decomposition of $G + F'$, which shows that $G + F'$ is trivially perfect. This contradicts the minimality of F.

(iv) Follows directly from *(i)* and *(ii)*: if $\ell > 0$, then $\ell > 1$ and every vertex of B has edges in G to all the different connected components of $D \setminus B$. □

10.2 Completion in subexponential time

As has been already mentioned, the following concept is crucial for our algorithm. Recall that when Ω is a set of vertices in a graph G, by m_Ω we mean the number of edges in $G[\Omega]$.

Definition 10.2 (Vital potential maximal clique)**.** Let (G, k) be an input instance to Trivially Perfect Completion. A vertex set $\Omega \subseteq V(G)$ is a *trivially perfect potential maximal clique* or simply *potential maximal clique*, if Ω is a maximal clique in some minimal trivially perfect completion of G. If moreover this trivially perfect completion contains at most k edges, then the potential maximal clique is called *vital*.

Observe that given a yes-instance (G, k) of Trivially Perfect Completion and a minimal completion F of size at most k, every maximal clique in $G + F$ is a vital potential maximal clique in G. Note also that in particular, any vital potential maximal clique contains at most k non-edges. The following definition will be useful:

Definition 10.3 (Fill number)**.** Let $G = (V, E)$ be a graph, F a completion of G, and $H = G + F$. We define the *fill number* of a vertex v, denoted by $\mathrm{fn}_H^G(v)$ as the number of edges incident to v in F.

Observation 10.4. *If (G, k) is a yes-instance of* Trivially Perfect Completion*, F a completion of G with $|F| \leq k$, and $H = G + F$, then there are at most $2\sqrt{k}$ vertices v in G such that $\mathrm{fn}_H^G(v) > \sqrt{k}$.*

It follows that for such a graph $G = (V, E)$ and every set $U \subseteq V$ such that $|U| > 2\sqrt{k}$, there is a vertex $u \in U$ with $\mathrm{fn}_H^G(u) \leq \sqrt{k}$. Any vertex u such that $\mathrm{fn}_H^G(u) \leq \sqrt{k}$ will be referred to as a *cheap* vertex. If u is not cheap, we call it *expensive*.

We are now ready to begin with the proof of Theorem 10. Our algorithm for Trivially Perfect Completion consists of three steps. We first compress, in polynomial time, the instance to an instance of size $O(k^3)$, then we enumerate all the (subexponentially many) vital potential maximal cliques in this new instance, and finally we do a dynamic programming procedure on these objects.

Step A. Kernelization. We start from one of the known polynomial kernelization algorithms for the problem. For a given input (G, k), we want to construct in polynomial time an equivalent instance (G', k'), where G' has $k^{O(1)}$ vertices and $k' \leq k$. Such a kernelization algorithm producing in time $O(kn^4)$ an equivalent instance with graph G' on $O(k^3)$ vertices was announced by Guo in [Guo07]. The existence of a larger polynomial kernel with $O(k^7)$ vertices was shown in Section 6.2. Let us note that for our algorithm any polynomial kernel will work fine.

From now on we assume that the input graph G has $O(k^3)$ vertices. (Replacing $O(k^3)$ by any other polynomial of k will not change our proof.) Without loss of generality, we will also assume that G is connected, since we can treat each connected component of G separately.

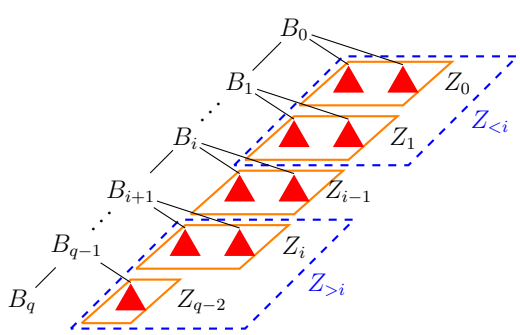
Step B. Enumeration. In this step, we describe an algorithm that in time $2^{O(\sqrt{k}\log k)}$ outputs a family $\mathcal{C}$ of vertex subsets of G such that

- the size of $\mathcal{C}$ is $2^{O(\sqrt{k}\log k)}$, and
- every vital potential maximal clique belongs to $\mathcal{C}$.

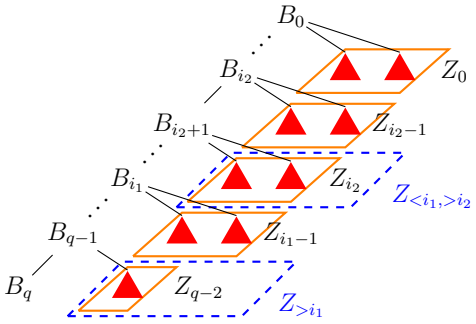
We identify four different types of vital potential maximal cliques. For each type i, $1 \leq i \leq 4$, we list a family $\mathcal{C}_i$ of $2^{O(\sqrt{k}\log k)}$ subsets containing all vital potential maximal cliques of that type. Finally, we set $\mathcal{C} = \mathcal{C}_1 \cup \cdots \cup \mathcal{C}_4$. We show that every vital potential maximal clique of (G, k) is of at least one of these types and that all objects of each type can be enumerated in $2^{O(\sqrt{k}\log k)}$ time.

Let Ω be a vital potential maximal clique of G. By the definition of Ω, there exists a minimal completion of G with at most k edges into a trivially perfect graph H such that Ω is a maximal clique in H. Let $(T = (V_T, E_T), \mathcal{B} = \{B_t\}_{t \in V_T})$ be the universal clique decomposition of H. Recall that by Lemma 2.27, Ω corresponds to a path $P_{rt} = B_{t_0} B_{t_1} \cdots B_{t_q}$ in T from the root $r = t_0$ to a leaf $t = t_q$. Then for the corresponding leaf block (B_t, D_t) with tail Q_t, we have that $\Omega = Q_t$. To simplify the notation, we use B_i for B_{t_i}.

Note that the algorithm knows neither the clique Ω nor the completed trivially perfect graph H. However, in the analysis we may partition all the vital potential maximal cliques Ω with respect to structural properties of Ω and H, and then provide simple enumeration rules which ensure that all vital potential maximal



(a) Illustration of the vital potential maximal cliques of Type 3. Z_i is the collection of vertices corresponding to bags below B_i but not below B_{i+1}.



(b) Illustration of vital potential maximal cliques of Type 4 . Indices i_1, i_2 are the two largest indices such that Z_{i_1} and Z_{i_2} contain cheap vertices.

Figure 10.1: Illustration of the different neighborhoods of the maximal clique that we use to find the maximal cliques of Types 3 and 4. The figure shows a universal clique decomposition of a completed graph, where $\Omega = B_0 \cup \ldots \cup B_q$ is a maximal clique. Observe that the leaf block is $L_q = (B_q, B_q)$ and that its tail is Ω.

cliques of each type are indeed enumerated. We now proceed to the description of the types and enumeration rules and refer to Figure 10.1 for a visualization of these concepts. Henceforth, whenever we are referring to cheap or expensive vertices, we mean being cheap/expensive with respect to a fixed completion to H.

Type 1. Potential maximal cliques of the first type are such that $|V \setminus \Omega| \leq 2\sqrt{k}$. The family $\mathcal{C}_1$ consists of all sets $W \subseteq V$ such that $|V \setminus W| \leq 2\sqrt{k}$. There are at most $(2\sqrt{k}+1) \cdot \binom{O(k^3)}{2\sqrt{k}}$ such sets, and using the fact that $\binom{a}{b} = a^{O(b)} = 2^{O(b \log a)}$, we enumerate all of them in time $2^{O(\sqrt{k} \log k)}$ by trying all vertex subsets of size at least $|V| - 2\sqrt{k}$. Thus every Type 1 vital potential maximal clique is in $\mathcal{C}_1$.

Type 2. By Lemma 2.27 *(i)*, we have that $\Omega = Q_t = N_H[v]$ for each vertex $v \in D_t = B_t$. Vital potential maximal cliques of the second type are those of the form $N_H[v]$ for some $v \in B_t$ where B_t is a leaf bag in the universal clique decomposition of H, and $|B_t| > 2\sqrt{k}$. We generate the family $\mathcal{C}_2$ as follows. Every set in $\mathcal{C}_2$ is of the form $W_1 \cup W_2$, where $W_1 = N_G[v]$ for some $v \in V$, and $|W_2| \leq \sqrt{k}$. There are at most $\binom{O(k^3)}{\sqrt{k}} \cdot O(k^3)$ such sets and they can be enumerated by computing for every vertex v the set $W_1 = N_G[v]$ and adding to each such set all possible subsets of size at most $\sqrt{k}$. Hence every Type 2 vital potential maximal clique is in $\mathcal{C}_2$.

Thus if Ω is not of Types 1 or 2, then $|V \setminus \Omega| > 2\sqrt{k}$ and for the corresponding leaf block we have $|B_t| \leq 2\sqrt{k}$. Since $|V \setminus \Omega| > 2\sqrt{k}$ it follows that $V \setminus \Omega$ contains at least one cheap vertex, i.e., a vertex with fill number at most $\sqrt{k}$.

We partition the nodes of T that are not on the path $B_0, B_1, \ldots, B_q$ into q disjoint sets $Z_0, Z_1, \ldots, Z_{q-1}$ according to the nodes of the path P_{rt}. Node $x \notin V(P_{rt})$ belongs to Z_i, $i \in \{0, \ldots, q-1\}$, if i is the largest integer such that t_i is

an ancestor of x in T. In other words, Z_i consists of bags of subtrees outside P_{rt} attached below t_i, see Figure 10.1a. For two integers p_1 and p_2, we will denote $B_{p_1,p_2} = \bigcup_{j=p_1}^{p_2} B_j$.

For the remaining two types of vital potential maximal cliques we distinguish cases depending on whether all cheap vertices in $V \setminus \Omega$ are located in exactly one set Z_i, or not. Recall that all vital potential maximal cliques for which $V \setminus \Omega$ does not contain any cheap vertex are already contained in Type 1.

Type 3. Vital potential maximal cliques Ω of the third type are the ones that do not belong to Type 1 or 2, but for which there exists an index $i \in \{0, 1, \ldots, q-1\}$ such that all cheap vertices of $V \setminus \Omega$ belong to Z_i. Since Ω is not of Type 1, Z_i is non-empty. Also, since Ω is not of Type 2, we have that $|B_q| \leq 2\sqrt{k}$. Let us denote $Z_{<i} = \bigcup_{j=0}^{i-1} Z_j$ and $Z_{>i} = \bigcup_{j=i+1}^{q-1} Z_j$ (see Figure 10.1a). By our assumption, we have that $Z_{<i}$ and $Z_{>i}$ contain only expensive vertices, and hence $|Z_{<i}|, |Z_{>i}| \leq 2\sqrt{k}$. Let u be any cheap vertex belonging to Z_i, and observe that the following equalities and inclusions are implied by Lemma 10.1 *(ii)*:

- $B_{0,i-1} = N_G(Z_{<i}) \subseteq \Omega$;
- $B_{i+1,q-1} \subseteq N_G(Z_{>i}) \subseteq \Omega$;
- $B_i \subseteq N_G(B_q \cup (N_G[Z_{>i}] \setminus N_H(u))) \subseteq \Omega$.

It follows that

$$\Omega = N_G(Z_{<i}) \cup N_G(Z_{>i}) \cup N_G\Big(B_q \cup \Big(N_G[Z_{>i}] \setminus N_H(u)\Big)\Big) \cup B_q. \tag{10.1}$$

Given (10.1), we may define family $\mathcal{C}_3$. Family $\mathcal{C}_3$ comprises all the sets that can be constructed as follows:

- Pick three disjoint sets $W_1, W_2, W_3 \subseteq V$ of size at most $2\sqrt{k}$ each. This corresponds to the choice of $Z_{<i}$, $Z_{>i}$ and B_q, respectively.
- Pick a vertex $v \in V$ and a set $A \subseteq V$ of size at most $\sqrt{k}$. This corresponds to the choice of u and fill-in edges adjacent to u. Let $N_v = N_G(v) \cup A$.
- Put the set $N_G(W_1) \cup N_G(W_2) \cup N_G(W_3 \cup (N_G[W_2] \setminus N_v)) \cup W_3$ into the family $\mathcal{C}_3$.

Observe that since $|V| = O(k^3)$, and since we enumerate all possible collections of four sets of size $O(\sqrt{k})$ and one vertex, the number of sets included in $\mathcal{C}_3$ is at most $\binom{O(k^3)}{O(\sqrt{k})}^4 \cdot O(k^3) = 2^{O(\sqrt{k}\log k)}$. Furthermore, this family can also be enumerated within the same asymptotic running time. From (10.1) it follows immediately that each vital potential maximal clique of Type 3 is contained in $\mathcal{C}_3$.

Type 4. Vital potential maximal cliques Ω of the fourth type are the ones that do not belong to Type 1 or 2, but there exist at least two indices i_1 and i_2 such that Z_{i_1} and Z_{i_2} both contain a cheap vertex. Let i_1, i_2 be the two largest such

indices, where $i_1 > i_2$. Let $Z_{<i_1,>i_2} = \bigcup_{j=i_2+1}^{i_1-1} Z_j$ and $Z_{>i_1} = \bigcup_{j=i_1+1}^{q-1} Z_j$. See Figure 10.1b for an illustration. By the maximality of i_1, i_2 we have that $Z_{<i_1,>i_2}$ and $Z_{>i_1}$ contain only expensive vertices, and hence that $|Z_{<i_1,>i_2}|, |Z_{>i_1}| \leq 2\sqrt{k}$. Again, since Ω is not of Type 2, we have that $|B_q| \leq 2\sqrt{k}$. Let $u_1 \in Z_{i_1}$ and $u_2 \in Z_{i_2}$ be two cheap vertices. Observe that the following equalities and inclusions are implied by Lemma 10.1 *(ii)*:

- $B_{0,i_2} = N_H(u_1) \cap N_H(u_2)$;
- $B_{i_2+1,i_1-1} \subseteq N_G(Z_{<i_1,>i_2}) \subseteq \Omega$;
- $B_{i_1+1,q-1} \subseteq N_G(Z_{>i_1}) \subseteq \Omega$;
- $B_{i_1} \subseteq N_G(B_q \cup (N_G[Z_{>i_1}] \setminus N_H(u_1))) \subseteq \Omega$.

It follows that

$$\begin{aligned}\Omega = \Big(N_H(u_1) \cap N_H(u_2)\Big) \cup N_G(Z_{<i_1,>i_2}) \cup N_G(Z_{>i_1}) \\ \cup N_G\Big(B_q \cup \Big(N_G[Z_{>i_1}] \setminus N_H(u_1)\Big)\Big) \cup B_q\,.\end{aligned} \tag{10.2}$$

Given (10.2), we may define the family $\mathcal{C}_4$. This family comprises all the sets that can be constructed as follows:

- Pick three disjoint sets $W_1, W_2, W_3 \subseteq V$ of size at most $2\sqrt{k}$ each. This corresponds to the choice of $Z_{<i_1,>i_2}$, $Z_{>i_1}$ and B_q, respectively.
- Pick two vertices $v_1, v_2 \in V$ and two sets $A_1, A_2 \subseteq V$, each of size at most $\sqrt{k}$. This corresponds to the choice of u_1 and u_2, and of the neighbors in H adjacent to u_1 and u_2. Let $N_{v_i} = N_G(v_i) \cup A_i$, for $i = 1, 2$.
- Put the set $(N_{v_1} \cap N_{v_2}) \cup N_G(W_1) \cup N_G(W_2) \cup N_G(W_3 \cup (N_G[W_2] \setminus N_{v_1})) \cup W_3$ into the family $\mathcal{C}_4$.

Observe that since $|V| = O(k^3)$, and by the same analysis as for Type 3, the number of sets included in $\mathcal{C}_4$ is at most $2^{O(\sqrt{k}\log k)}$, and that this family can be enumerated within the same asymptotic running time. From (10.2) it follows immediately that each vital potential maximal clique of Type 4 is contained in $\mathcal{C}_4$.

Summarizing, every vital potential maximal clique of Types 1, 2, 3, and 4 is included in the family $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$, and $\mathcal{C}_4$, respectively. Since every vital potential maximal clique is of Types 1, 2, 3, or 4, by taking $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3 \cup \mathcal{C}_4$ we can infer the following lemma that formalizes the result of Step B.

Lemma 10.5 (Enumeration lemma)**.** *Let (G, k) be an instance of* Trivially Perfect Completion *such that $|V(G)| = O(k^3)$. Then in time $2^{O(\sqrt{k}\log k)}$, we can construct a family $\mathcal{C}$ consisting of $2^{O(\sqrt{k}\log k)}$ subsets of $V(G)$ such that every vital potential maximal clique of (G, k) is in $\mathcal{C}$.*

Step C. Dynamic programming. We first give an intuitive idea of the dynamic programming procedure: We start off by assuming that we have the family $\mathcal{C}$ containing all vital potential maximal cliques of (G, k). We start by generating in time $2^{O(\sqrt{k}\log k)}$ a family $\mathcal{F}$ of pairs (X, Y), where $X, Y \subseteq V(G)$, such that for every minimal completion F of size at most k, and the corresponding universal clique decomposition $(T, \mathcal{B})$ of $H = G + F$, it holds that every block (B, D) is in $\mathcal{F}$, and the size of $\mathcal{F}$ is $2^{O(\sqrt{k}\log k)}$. (See Definition 2.26 for the definition of a block.)

The construction of $\mathcal{F}$ is based on the following observations about blocks and vital potential maximal cliques: Let G be a graph, F a minimal completion and $L = (B, D)$ a block of the universal clique decomposition of $H = G + F$, where H is not a complete graph, with Q being its tail. Then the following hold, as we prove later:

- If L is a leaf block, then $B = \Omega_1 \setminus \Omega_2$ for some vital potential maximal cliques Ω_1 and Ω_2, and $D = B$.
- If L is the root block, then the tail of L is B, $B = \Omega_1 \cap \Omega_2$ for some vital potential maximal cliques Ω_1 and Ω_2, and $D = V$.
- If L is an internal block, then Q is the intersection of two vital potential maximal cliques Ω_1 and Ω_2 of G, $B = Q \setminus \Omega_3$ for some vital potential maximal clique Ω_3, and D is the connected component of $G - (Q \setminus B)$ containing B.

From this observation, we can conclude that by going through all triples $\Omega_1, \Omega_2, \Omega_3$ of elements of $\mathcal{C}$, we can compute the set $\mathcal{F}$ consisting of all blocks (B, D) of minimal completions. We now define the value $\mathrm{dp}(B, D)$ as follows: $\mathrm{dp}(B, D)$ is equal to the minimum number of edges needed to be added to $G[D]$ to make it a trivially perfect graph with B being the universal clique contained in the root of its universal clique decomposition, unless this minimum number is larger than k; in this case we put $\mathrm{dp}(B, D) = +\infty$. We later derive recurrence equations that enable us to compute all the relevant values of $\mathrm{dp}(\cdot, \cdot)$ using dynamic programming. Finally, the minimum cost of completing G to a trivially perfect graph is equal to $\min_{(B,V(G))\in\mathcal{F}} \mathrm{dp}(B, V(G))$. If this minimum is equal to $+\infty$, then no completion of size at most k exists and we can conclude that (G, k) is a no-instance.

We now proceed to a formal proof of the correctness of the dynamic programming procedure. Suppose that we have the family $\mathcal{C}$ containing all vital potential maximal cliques of (G, k). We start by generating in time $2^{O(\sqrt{k}\log k)}$ a family $\mathcal{F}$ of pairs (X, Y), where $X, Y \subseteq V$, where $V = V(G)$, such that

- for every minimal completion H that adds at most k edges, every block (B, D) of the universal clique decomposition of H belongs to $\mathcal{F}$, and
- the size of $\mathcal{F}$ is $2^{O(\sqrt{k}\log k)}$.

The construction of $\mathcal{F}$ is based on the following lemmata.

Lemma 10.6. *Let G be a graph, F a minimal completion of G of size at most k, and (B, D) a non-leaf and non-root block of the universal clique decomposition of $H = G + F$, with Q being its tail. Then*

(i) Q is the intersection of two vital potential maximal cliques Ω_1 and Ω_2 of G,

(ii) $B = Q \setminus \Omega_3$ for some vital potential maximal clique Ω_3, and

(iii) D is the connected component of $G - (Q \setminus B)$ containing B.

Proof. (i) Consider two connected components D_1 and D_2 of $H[D \setminus B]$ and let Ω_1' and Ω_2' be maximal cliques in D_1 and D_2. Observe that $\Omega_1 = \Omega_1' \cup Q$ and $\Omega_2 = \Omega_2' \cup Q$ are maximal cliques in H. By definition, Ω_1 and Ω_2 are vital potential maximal cliques in G, and $\Omega_1 \cap \Omega_2 = Q$.

(ii) Let $\hat{L} = (\hat{B}, \hat{D})$ be the parent block of (B, D). Since $\hat{L}$ is not a leaf-block, $\hat{L}$ has at least two children and thus there is a block $(B', D') \neq (B, D)$ which is also a child of $\hat{L}$. By the previous argument, $\hat{Q}$, the tail of $\hat{L}$ is exactly $\hat{Q} = \Omega_1 \cap \Omega_3$ for some vital potential maximal clique Ω_3. It follows that $B = Q \setminus \Omega_3$.

(iii) It follows from Lemma 10.1 that $G[D]$ is connected. Then it follows immediately that D is the unique connected component of $G-(Q\setminus B)$ containing B. □

Lemma 10.7. *Let G be a graph, F a minimal completion of G of size at most k, and $L = (B, D)$ a leaf block of the universal clique decomposition of $H = G + F$. If H is not a complete graph, then*

(i) $B = \Omega_1 \setminus \Omega_2$ for some vital potential maximal cliques Ω_1 and Ω_2, and

(ii) $D = B$.

Proof. (i) Let $\hat{L} = (\hat{B}, \hat{D})$ be the parent block of L, which exists since L is not the root block. Let $L' = (B', D')$ be a child of $\hat{L}$ which is not L. If $L' = (B', D')$ is a leaf, then set $L'' = L'$, and if not, then let $L'' = (B'', D'')$ be a leaf which has L' as an ancestor. The blocks L' and L'' exist since $\hat{L}$ is not a leaf. Furthermore, let $\hat{Q}$ be the tail of $\hat{L}$, and let $\Omega_1 = N_H[B]$ and $\Omega_2 = N_H[B'']$; Ω_1 and Ω_2 are then two maximal cliques in H. From similar arguments as above we get that $\hat{Q} = \Omega_1 \cap \Omega_2$ and hence that $B = \Omega_1 \setminus \Omega_2$.

(ii) This follows immediately from Lemma 2.27. □

Lemma 10.8. *Let G be a connected graph, F a minimal completion of G of size at most k, and $L = (B, D)$ the root block of the universal clique decomposition of $H = G + F$. If H is not a complete graph, then*

(i) the tail of L is B,

(ii) $B = \Omega_1 \cap \Omega_2$ for some vital potential maximal cliques Ω_1 and Ω_2, and

(iii) $D = V$.

Proof. (i) By definition, the tail is the collection of vertices from B to the root. Since L is a root block, the tail is B itself.

(ii) This follows in the same manner as in the proof of Lemma 10.6 *(i)*, since B is the tail of block L.

(iii) From the definition of universal clique decompositions we have that D is the connected component of $H[V \setminus (Q \setminus B)]$ containing B. But since $Q \setminus B = \emptyset$, we get that D is the connected component of H containing B. Now since H is connected, the result follows. □

By making use of Lemmata 10.6–10.8, one can construct the required family $\mathcal{F}$ by going through all possible triples of elements of $\mathcal{C}$. The size of $\mathcal{F}$ is at most $|\mathcal{C}|^3 = 2^{O(\sqrt{k}\log k)}$ and the running time of the construction of $\mathcal{F}$ is $2^{O(\sqrt{k}\log k)}$. Note here that by Lemma 10.1 *(iii)* and the fact that G is connected, we may discard from $\mathcal{F}$ every pair (B, D) where $G[D]$ is not connected.

For every pair $(X, Y) \in \mathcal{F}$, with $X \subseteq Y \subseteq V$, we define $\mathrm{dp}\,(X, Y)$ to be the minimum number of edges required to add to $G[Y]$ to obtain a trivially perfect graph where X is the maximal universal clique; If this minimum value exceeds k, we define $\mathrm{dp}\,(X, Y) = +\infty$. Thus, to compute an optimal solution, it is sufficient to go through the values $\mathrm{dp}\,(X, Y)$, where $(X, Y) \in \mathcal{F}$ with $Y = V$. In other words, to compute the size of a minimum completion we find

$$\min_{(X,V)\in\mathcal{F}} \mathrm{dp}\,(X, V), \tag{10.3}$$

and if this value is $+\infty$, then the size of a minimum completion exceeds k.

In the following, for a subset of vertices A we write m_A to denote the number of edges inside A, i.e., $m_A = |E(A)|$. We compute (10.3) by making use of dynamic programming over elements of $\mathcal{F}$. For every pair $(X, Y) \in \mathcal{F}$ which can be a leaf block for some completion, i.e., for all pairs with $X = Y$, we put

$$\mathrm{dp}\,(X, X) = \binom{|X|}{2} - m_X.$$

Of course, if the computed value exceeds k, then we put $\mathrm{dp}\,(X, X) = +\infty$.

For $(X, Y) \in \mathcal{F}$ with $X \subsetneq Y$, if (X, Y) is a block of some minimal completion H, then in H, we have that X is a maximal universal clique in $H[Y]$, every vertex of X is adjacent to all vertices of $Y \setminus X$ and the number of edges in $H[Y \setminus X]$ is the sum of the number of edges in the connected components of $H[Y \setminus X]$. By Lemma 10.1, the vertices of every connected component Y' of $H[Y \setminus X]$ induce a connected component in $G[Y \setminus X]$. Observe that each connected component Y' of $H[Y \setminus X]$ corresponds to a block (X', Y') in the decomposition of H. Now since $\mathcal{F}$ contains all blocks of minimal trivially perfect completions it follows that $(X', Y') \in \mathcal{F}$.

Now for $(X, Y) \in \mathcal{F}$ we use $m_{X,Y\setminus X} = |E(X, Y \setminus X)|$ to denote the number of edges between X and $Y \setminus X$ in G. Let C be the set of connected components

of $G[Y \setminus X]$. Then we compute, in order of increasing size of Y,

$$\mathrm{dp}\,(X,Y) = \binom{|X|}{2} - m_X + |X| \cdot |Y \setminus X| - m_{X,Y\setminus X} + \sum_{G[Y']\in C} \min_{(X',Y')\in\mathcal{F}} \mathrm{dp}\,(X',Y')\,.$$

Again, if the value on the right hand side exceeds k, then we set $\mathrm{dp}\,(X,Y) = +\infty$.

Since the cardinality of Y' is less than $|Y|$, and blocks are processed in increasing cardinality of Y, the value for $\mathrm{dp}\,(X',Y')$ has already been calculated when it is needed to compute $\mathrm{dp}\,(X,Y)$.

The running time required to compute $\mathrm{dp}\,(X,Y)$ is, up to a polynomial factor in k, proportional to the number of sets $(X',Y') \in \mathcal{F}$, which is $O(|\mathcal{F}|)$. Thus the total running time of the dynamic programming procedure is, up to a polynomial factor in k, proportional to $O(|\mathcal{F}|^2)$, and hence (10.3) can be computed in time $2^{O(\sqrt{k}\log k)}$. This concludes Step C and the proof of Theorem 10.

Chapter 11

Pseudosplit graphs

The pseudosplit graphs are exactly the $\{2K_2, C_4\}$-free graphs. Since $2K_2 = \overline{C_4}$, by Fact 2.7, the problem Pseudosplit Completion is equivalent to Pseudosplit Deletion. The problem Split Editing [HS81] is well known to be solvable in polynomial time. Essentially, by guessing the induced C_5, so is Pseudosplit Editing. We give a short proof for this and motivate it more later.

11.1 Polynomial time editing

Recall that in the problem Split Editing, we are given a graph $G = (V, E)$ and an integer $k \geq 0$ and asked whether there exists a split editing set F for G of size at most k.

The answer is given by the formula $\sigma(G)$, the *splittance* [HS81] of G, defined by

$$\sigma(G) = \frac{1}{2}\left(k(k-1) - \sum_{i=1}^{k} d_i + \sum_{i=k+1}^{n} d_i\right), \tag{11.1}$$

where d_i denotes the degree of the ith highest degree vertex, and $k = \max\{i \mid 1 \leq i \leq n, d_i \geq i-1\}$. Hammer and Simeone [HS81] showed that the size of the minimum split editing set F for a graph G is $|F| = \sigma(G)$. This shows that Split Editing, with standard sorting arguments, can be computed in linear time, by computing the right hand side of Equation (11.1).

In Pseudosplit Editing, we are given a graph G and an integer $k \geq 0$ and asked whether there is a pseudosplit editing set of size at most k, i.e., does there exist an edge set F such that $G \triangle F$ is a pseudosplit graph and $|F| \leq k$? Since a pseudosplit graph is either a split graph, or an imperfect pseudosplit graph, and that the split editing problem is solvable in linear time, we may consider the complexity of editing a graph to a so-called *imperfect pseudosplit graph*, a graph in which the C_5 is present. We refer to this problem as Imperfect Pseudosplit Editing.

Define $\psi(G)$ to be the size of a minimum imperfect pseudosplit editing set. We aim to show that $\psi(G)$ can be computed in the same manner as $\sigma(G)$, only taking into consideration the induced C_5. Finally, we conclude that Pseudosplit

EDITING can be computed by taking the minimum of the two parameters, namely $\min(\sigma(G), \psi(G))$.

Proposition 11.1. *For a proper pseudosplit graph G, we have that $\sigma(G) \leq 2$.*

Proof. Let C, S, I be a pseudosplit partition of $V(G)$. Suppose $s_1s_2s_3s_4s_5s_1$ is the five-cycle $G[S] \cong C_5$. Let $F = \{s_3s_4, s_4s_5\}$. Now it suffices to observe that $G \triangle F$ is a split graph on partition $C \cup \{s_1, s_2\}, I \cup \{s_3, s_4, s_5\}$. □

Lemma 11.2. *Let F_ψ be a pseudosplit editing set of a graph G with pseudosplit partition C, S, I. If one of the vertices in S is incident with at least two edges of C or of I in F_ψ, then there is a split editing set F_σ of size at most $|F_\psi|$.*

Proof. Let F_ψ be pseudosplit edit set for which $G \triangle F_\psi$ has partition C, S, I. Let $s_1s_2s_3s_4s_5s_1$ denote the five-cycle S and suppose without loss of generality that s_1 is incident with two edges $e_1 = s_1c_1$ and $e_2 = s_1c_2$ with $c_1, c_2 \in C$. Then $F_\sigma = F_\psi \setminus \{e_1, e_2\} \cup \{s_1s_5, s_1s_2\}$ is a split edit set for which $G \triangle F_\sigma$ is a split graph on partition $C \cup \{s_3, s_4\}, I \cup \{s_1, s_2, s_5\}$. The proof of the case when s_1 is incident with at least two edges of I is similar, except we remove s_3s_4 and s_4s_5, and the split partition will be $C \cup \{s_1, s_2\}, I \cup \{s_3, s_4, s_5\}$. □

From Lemma 11.2, we can immediately conclude the following:

Corollary 11.3. *Let G be a graph such that $\psi(G) < \sigma(G)$. Then an optimal pseudosplit edit set F_ψ with partition C, S, I has the property that every $s \in S$ is incident with at most two edges of $V(G) \setminus S$ in F_ψ.*

The following lemma essentially shows that if $\psi(G) < \sigma(G)$, then the set S already induces a C_5 in G.

Lemma 11.4. *If F_ψ is a pseudosplit edit set which contains an edge e with both endpoints in S, then there is a split edit set $F_\sigma = F_\psi \setminus \{e\} \cup \{e'\}$, where e' also has both endpoints in S.*

Proof. There are two cases, either e is an added edge, without loss of generality s_1s_2, or e is a deleted edge, w.l.o.g. s_1s_3. If $e = s_1s_2$ is an added edge, then let $e' = s_2s_3$ and the claim follows by observing that $G \triangle F_\sigma$ has split partition $C \cup \{s_4, s_5\}, I \cup \{s_1, s_2, s_3\}$. If $e = s_1s_3$ is the deleted edge, then we let $e' = s_4s_5$ and $G \triangle F_\sigma$ has split partition $C \cup \{s_1, s_2, s_3\}, I \cup \{s_4, s_5\}$. □

Corollary 11.5. *Let G be a graph such that $\psi(G) < \sigma(G)$. Then it follows from Corollary 11.3 and Lemma 11.4 that there is a set $S \subseteq V(G)$ such that $s \in S$ is incident with at most two edges of $V \setminus S$ and furthermore $G[S] \cong C_5$.*

Lemma 11.6. *Let G be a graph and F_ψ a pseudosplit edit set with C, S, I the pseudosplit partition of $G \triangle F_\psi$. If there are more than two edges incident with S in F_ψ, then there is a split edit set F_σ of size at most F_ψ.*

Algorithm 2 Algorithm solving Pseudosplit Editing.

On input (G, k),

1. Compute $\sigma(G)$. If $\sigma(G) > k + 2$, by Proposition 11.1 we return no. If $\sigma(G) \leq k$, we return yes.
2. Try every induced C_5 $S = \{s_1, \ldots, s_5\}$ in $V(G)$. Let $C' = N(S)$ and $I' = V(G) \setminus (C' \cup S)$. For each u, v of $C' \cup I'$, we compute $k_{C,S,I}$, where C and I are the possible ways of distributing u and v in C and I. Given such a partition C, S, I, we return true if $k_{C,S,I} \leq k$.
3. Return no (Corollary 11.5) if no induced C_5 satisfied the above inequality.

Proof. Let e_1, e_2 and e_3 be the edges in F_ψ incident with vertices in S and let us assume by Corollary 11.5 that they are not (a) having both endpoints in S and (b) not all incident with the same vertex in S. Suppose that s_i and s_j are two vertices in S incident with two of the edges, without loss of generality e_1 and e_2. Let $e_1 = s_i v$ and $e_2 = s_j u$. We distinguish between two cases, *(i)* v and u are (w.l.o.g.) contained in I or *(ii)* $v \in C$ and $u \in I$.

For *(i)*, let e be an edge in $G[S]$ not incident with s_i nor s_j and let e' be the non-edge in $G[S]$ that is needed to add to make s_i and s_j be contained in a triangle. Now, $F_\sigma = F_\psi \setminus \{e_1, e_2\} \cup \{e, e'\}$ is a split edit set. For *(ii)*, we follow the same strategy, but we want s_i to go to I and s_j to go to C. So we let e be a non-edge in $G[S]$ that will make s_j be contained in a triangle without s_i, and we delete the edge needed to make $G[S]$ into a bull[1]. It follows that $F_\sigma = F_\psi \setminus \{e_1, e_2\} \cup \{e, e'\}$ is a split edit set. □

Lemma 11.7. *Let G be a graph with $\psi(G) < \sigma(G)$ and let F_ψ be a minimum edit set with $C, S, I = (G \triangle F_\psi)$. Then for at least four vertices $s_1, \ldots, s_4 \in S$, we have that $N_G(s_i) \setminus S = C$.*

Proof. We know that at most two edges of F_ψ are incident with S, and if there are two edges $e_1, e_2 \in F_\psi$ incident with S, then e_1, e_2 are both incident with the same vertex $s \in S$. Hence for $S \setminus \{s\}$, we have that $N_G(S \setminus \{s\}) = C$. □

Definition 11.8. Let G be a graph and C, S, I a partition of the vertex set with the restriction that $G[S] \cong C_5$. We define

$$k_{C,S,I} = (|C|(|C| - 1))/2 - |E(C)| + |\overline{E}(C, S)| + |E(I)| + |E(S, I)|,$$

that is, $k_{C,S,I}$ is the cost of editing G into an imperfect pseudosplit graph with partition C, S, I.

The following theorem and the correctness of Algorithm 2 is straightforwards from the above exposition.

[1]See Table 2.1 for a description of the bull.

Theorem 11. *The pseudosplit number of a graph G can be computed in time $O(n^8)$.*

As mentioned earlier, and as Yannakakis also stated, we are far from understanding the nature of the complexity of edge modification problems. One interesting open problem is for which finite sets $\mathcal{H}$ the problems $\mathcal{H}$-free Completion, $\mathcal{H}$-free Deletion, and $\mathcal{H}$-free Editing are NP-complete. In the latter case, we have somehow "sharpened" the knowledge here by stating that $\{2K_2, C_4\}$-free Editing, in addition to $\{2K_2, C_4, C_5\}$-free Editing, is polynomial time solvable.

In addition to this observation on classical complexity, it partially refutes an open problem posed by Cai and Cai [CC15, Problem 7.6] on kernelization complexity. They ask the following: Suppose that neither H_1-free Editing nor H_2-free Editing admit a polynomial kernel (under the assumption NP $\not\subseteq$ coNP/poly), does it hold that $\{H_1, H_2\}$-free Editing is incompressible? In our case, C_4-free Editing, which is equivalent to $2K_2$-free Editing, is incompressible, by their work. However, we have just seen that $\{2K_2, C_4\}$-free Editing is even solvable in polynomial time.

The answer is only partial, though, since they simultaneously ask for the edge deletion version. This is to the best of the author's knowledge still open. However, it seems likely that here one can also simply provide a polynomial kernel for Pseudosplit Deletion, and thus refute their conjecture.

11.2 Subexponential time completion

In this section we show that Pseudosplit Completion can be solved by first applying a polynomial-time and parameter-preserving preprocessing routine, and then using the subexponential time algorithm of Ghosh et al. [GKK$^+$15] for Split Completion. The NP-completeness of the problem follows from the NP-completeness of Split Completion.

In order to ease the argumentation regarding minimal completions, we call a split partition (C, I) *I-maximal* if there is no vertex $v \in C$ such that $(C \setminus \{v\}, I \cup \{v\})$ is a split partition. Our algorithm uses the subexponential algorithm of Ghosh et al. [GKK$^+$15] for Split Completion as a subroutine. We therefore need the following result (which we also proved in Chapter 9, Corollary 9.4):

Proposition 11.9 ([GKK$^+$15])**.** Split Completion *is solvable in subexponential parameterized time $2^{O(\sqrt{k}\log k)} + \mathrm{poly}(n)$.*

Formally, in this section we prove the following theorem:

Theorem 12. Pseudosplit Completion *is solvable in subexponential parameterized time $2^{O(\sqrt{k}\log k)} \cdot \mathrm{poly}(n)$.*

The algorithm whose existence is asserted in Theorem 12 is given as Algorithm 3. We now proceed to prove that this algorithm is correct, and that its running time

Algorithm 3 Algorithm solving Pseudosplit Completion.

1. Use the algorithm from Proposition 11.9 to check in time $2^{O(\sqrt{k}\log k)} \cdot \mathrm{poly}(n)$ if (G, k) is a yes-instance of Split Completion. If (G, k) is a yes-instance of Split Completion, then return that (G, k) is a yes-instance of Pseudosplit Completion. Otherwise we complete to an imperfect pseudosplit graph.

2. For each $S = \{x_1, x_2, \ldots, x_5\} \subseteq V(G)$ such that there is a supergraph $G_S \supseteq G[S]$ and $G_X \cong C_5$, we construct an instance (G', k') to Split Completion from (G, k) as follows:

   (a) Let $k' = k + |E(G[X])| - 5$.

   (b) Add all the possible edges between vertices of S, so that S becomes a clique.

   (c) Add a set A of $k + 2$ vertices to G.

   (d) Add every possible edge between A and $N_G[S]$.

3. Use Proposition 11.9 to check if (G', k') is a yes-instance of Split Completion. If (G', k') is a yes-instance of Split Completion, then return that (G, k) is a yes-instance of Pseudosplit Completion.

4. If for no set S the answer yes was returned, then return no.

on input (G, k) is $2^{O(\sqrt{k}\log k)} \cdot \mathrm{poly}(n)$. In the following we adopt the notation from Algorithm 3.

As in the algorithm, we denote by S the set of five vertices which will be used as the set inducing a C_5 (we try all possible subsets; note that their number is bounded by $O(n^5)$). Note here that since $G[S]$ admits a supergraph isomorphic to a C_5, it follows that $|E(G[S])| \leq 5$ and, consequently, $k' \leq k$. Similarly, by A we denote the set of $k+2$ vertices we add that are adjacent only to $N_G[S]$. Intuitively, this set will be used to force that in any minimal split completion of size at most k it holds that $N_G[S] \subseteq C$. From now on G' is the graph as in the algorithm, that is, G' is constructed from G by making S into a clique, adding vertices A and all the possible edges between A and $N_G[S]$.

The following lemma will be crucial in the proof of the correctness of the algorithm.

Lemma 11.10. *Assume that F is a minimal split completion of G' of size at most k', and let (C, I) be an I-maximal split partition of $G' + F$. Then:*

(i) $N_G[S] \subseteq C$,

(ii) $A \subseteq I$,

(iii) no edge of F has an endpoint in A,

(iv) $C \setminus S$ *is fully adjacent to* S *in* $G' + F$*, and*

(v) $I \setminus A$ *is fully non-adjacent to* S *in* $G' + F$*.*

Proof. (i) Aiming towards a contradiction, suppose that some $v \in N_G[S]$ is in I. Since $A \subseteq N(v)$, we must then have that $A \subseteq C$. However, since A is independent in G', this demands adding at least $\binom{k+2}{2} > k'$ edges.

(ii) Aiming towards a contradiction, assume that $A \cap C \neq \emptyset$. Since $N_{G'}(A) \subseteq C$ and A is independent in G', it follows that $G' + F'$, where $F' = F \setminus (A \times A)$ is also a split graph with partition (C', I'), where $C' = C \setminus A$ and $I' = I \cup (A \cap C)$. Since $F' \subseteq F$ holds, we have that either $|F'| < |F|$ which contradicts the minimality of F, or that $F' = F$ which contradicts the assumption that partition (C, I) was I-maximal.

(iii) Suppose that there is an edge $e \in F$ incident with a vertex of A. Since $A \subseteq I$, we infer that $F \setminus \{e\}$ is still a split completion, which contradicts the minimality of F.

(iv) C is a clique in $G' + F$ and $S \subseteq C$, so this holds trivially.

(v) Suppose for the sake of a contradiction that some $v^i \in I \setminus A$ is adjacent to some $v^x \in S$. Since $N_G[S] \subseteq C$, we have that $v^i v^x \in F$. But then $F \setminus \{v^i v^x\}$ is also a split completion, which contradicts the minimality of F. □

The correctness of the algorithm is implied by the following lemma:

Lemma 11.11. *The instance* (G, k) *is a yes-instance of* Pseudosplit Completion *if and only if Algorithm 3 returns yes on input* (G, k)*.*

Proof. In the forwards direction, let (G, k) be a yes-instance for Pseudosplit Completion. Observe that (G, k) is a yes-instance for Split Completion if and only if our algorithm returns yes in the first test. We therefore assume that G has to be completed to an imperfect pseudosplit graph.

Let F_0 be a completion set with $|F_0| \leq k$ such that $G_0 = G + F_0$ is an imperfect pseudosplit graph. Let (C, I, X) be the pseudosplit partition of $G + F_0$; hence $G_0[S]$ is isomorphic to a C_5. We claim that the algorithm will return yes when considering the set S in the second step; let then G' be the graph constructed in the second step of the algorithm, starting with the set S. Let F be equal to F_0 with all the edges of $G_0[S]$ that were not present in $G[S]$ removed; note that $|F| = |F_0| + |E(G[S])| - 5 \leq k'$. We claim that $G' + F$ is a split graph with split partition $(C \cup S, I \cup A)$. Indeed, since $G'[S]$ is a clique, S is fully adjacent to C in $G_0 \subseteq G' + F$, and C is a clique in $G_0 \subseteq G' + F$, it follows that $C \cup S$ is a clique in $G' + F$. On the other hand, $I \cup A$ is independent in G' and all the edges of F have at least one endpoint belonging to $C \cup S$, so $I \cup A$ remains independent in $G' + F$. As a result $G' + F$ is a split graph, and so the algorithm will return yes after the application of Proposition 11.9 in the third step.

In the reverse direction, assume that Algorithm 3 returns yes on input (G, k). If it returned yes already after the first test, then G may be completed into a

split graph by adding at most k edges, so in particular (G, k) is a yes-instance of Pseudosplit Completion. From now on we assume that the algorithm returned yes in the third step. More precisely, for some set S the application of Proposition 11.9 has found a minimal completion set F of size at most k' such that $G' + F$ is a split graph, with I-maximal split partition (C, I).

By Lemma 11.10 we have that *(i)* $N_G[S] \subseteq C$, *(ii)* $A \subseteq I$, *(iii)* no edge of F has an endpoint in A, *(iv)* $C \setminus S$ is fully adjacent to S in $G' + F$, and *(v)* $I \setminus A$ is fully non-adjacent to S in $G' + F$. By the choice of S, there exists a supergraph G_S of $G[S]$ such that $G_S \cong C_5$. Let now F_0 be equal to F with all the edges of G_S that were not present in $G[S]$ included. Observe that $|F_0| \leq k$ and that by *(iii)* F_0 contains only edges incident with vertices of G. Consider now the partition $(C \setminus S, I \setminus A, S)$ of $V(G + F_0)$. Since (C, I) was a split partition of $G' + F$, it follows that $C \setminus S$ is a clique in $G + F_0$ and $I \setminus A$ is an independent set in $G + F_0$. Moreover, from *(iv)* and *(v)* it follows that S is fully adjacent to $C \setminus S$ in $G + F_0$ and fully non-adjacent to $I \setminus A$ in $G + F_0$. Finally, the graph induced by S in $G + F_0$ is $G_S \cong C_5$. By Proposition 2.19 we infer that $G + F_0$ is a pseudosplit graph, and so the instance (G, k) is a yes-instance of Pseudosplit Completion. □

As for the time complexity of the algorithm, we try sets of five vertices for S, which is $O(n^5)$ tries. For each such guess, we construct the graph G', which has $n + k + 2$ vertices. Since $k' \leq k$, by Proposition 11.9 solving Split Completion requires time $2^{O(\sqrt{k} \log k)} \cdot \mathrm{poly}(n)$, both in the first and the third steps of the algorithm. Thus the total running time of Algorithm 3 is $2^{O(\sqrt{k} \log k)} \cdot \mathrm{poly}(n)$.

Chapter 12

Bicluster graphs and starforests

12.1 Subexponential time starforest editing

A first natural step in handling modification problems related to bicluster graphs is modification towards the subclass of bicluster graphs called starforests. Recall that a graph is a starforest if it is a forest of stars, or, a bicluster where every biclique has one side of size exactly one.

STARFOREST EDITING parameterized by k

Input:	A graph $G = (V, E)$ and a non-negative integer k.
Question:	Is there a set of at most k edges F such that $G \triangle F$ is a disjoint union of stars?

The problem where we only allow to *delete* edges is referred to as STARFOREST DELETION. These two problems can easily be observed to be equivalent; Adding an edge to a forbidden induced subgraph will create one of the other forbidden subgraphs, or simply put, it never makes sense to add an edge.

In Chapter 16 we show that this problem is NP-hard, and that it is not solvable in time $2^{o(k)}\,\mathrm{poly}(n)$ unless the exponential time hypothesis fails.

Multivariate analysis. Since no subexponential algorithm is possible under ETH, we introduce a secondary parameter, p, which bounds the number of connected components in a solution graph. This has previously been done with success in the CLUSTER EDITING problem [FKP+14]. Hence, we define the following multivariate variant of the above problem.

p-STARFOREST EDITING parameterized by p, k

Input:	A graph $G = (V, E)$ and a non-negative integer k.
Question:	Is there a set F of edges of size at most k such that $G \triangle F$ is a disjoint union of exactly p stars?

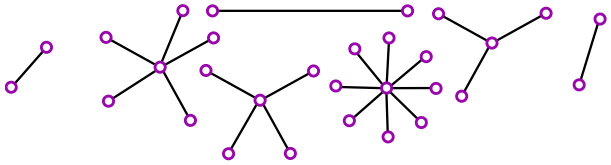


Figure 12.1: A starforest

Observe that this problem is *not* the same as p-Starforest Deletion since we might need to merge stars to achieve the desired value p for the number of connected components. In Section 16.2 (Theorem 24) we show that the problem is W[1]-hard parameterized by p alone, and that we therefore need to parameterize on both p and k.

Lemma 12.1. *Let (G, k) be input to p-Starforest Editing. If (G, k) is a yes-instance, there can be at most $p + 2k$ vertices with degree at least 2.*

Proof. Suppose $H = G \triangle F$ is a disjoint union of p stars with $|F| \leq k$. Let C be the set of p centers. Now, $V(H) \setminus C$ is a set of leaves of which at most $2k$ can be incident to F in G. Since all other leaves must already have degree one in G, the claim follows. □

The following bound will be key to obtain the subexponential running time.

Proposition 12.2 ([FKP+14])**.** *If a and b are non-negative integers, then $\binom{a+b}{a} \leq 2^{2\sqrt{ab}}$.*

Lemma 12.3. *Given a graph G and a vertex set S, we can compute in linear time $O(n + m)$ an optimal editing set F such that $G \triangle F$ is a starforest, when restricted to have S as the set of centers in the solutions.*

Proof. Observe that we need to delete every edge whose endpoints either lie both inside S or both outside of S. What remains is a bipartite graph with S being one side of the bipartition. To complete the editing, for every vertex $v \in V \setminus S$, with $\deg(v) > 1$, we delete all but one edge, and for every isolated vertex, we arbitrarily attach it to some vertex of S. It is easy to see that this solution is optimal. □

We now describe an algorithm which solves the problem p-Starforest Editing in time $O(2^{3\sqrt{pk}} + n + m)$.

The algorithm. Let (G, k) be an instance of p-Starforest Editing. If the number of vertices of degree at least two is greater than $p + 2k$, we say no in accordance with Lemma 12.1. Otherwise we split the graph into G_1 and G_2 as follows: Let $X \subseteq V(G)$ be the collection of vertices contained in connected components of size one or two, i.e. $G[X]$ is a collection of isolated vertices and edges. Let $G_1 = G[X]$ and $G_2 = G[V(G) \setminus X]$. Clearly, there are no edges going

out of X in G. We will treat G_1, G_2 as (almost) independent subinstances by guessing the budgets $k_1 + k_2 = k$ and the number of components in their respective solutions $p_1 + p_2 = p$. The only time we cannot treat them as independent instances is when p_1 or p_2 is zero; Let p_i^* be the number of stars completely contained in G_i in an optimal solution. If both $p_i^* > 0$, then there always exist an optimal solution that does not add any edge between G_1 and G_2.

Solving (G_1, k_1) with p_1 components: Assume G_1 contains s isolated edges and t isolated vertices, with $p_1 > 0$. If $|V(G_1)| < p_1$, we immediately say no, since we need exactly p_1 connected components. Depending on the values of s and t, we execute the following operations as long as the budget k_1 is positive. If $s \leq p_1$ and $s + t \leq p_1$, we have too few stars, and we arbitrarily delete edges to increase the number of connected components to p_1.

If $s = 0$ we turn the isolated vertices arbitrarily into p_1 stars. Otherwise, fix an arbitrary endpoint c of an isolated edge. Assume that $s \leq p_1$: then we connect enough isolated vertices to c such that the number of stars is p_1. Finally, if $s > p_1$, we first dissolve $s - p_1$ edges and continue as in the previous case. It is easy to check that the above solutions are optimal.

Solving (G_2, k_2) with p_2 components: By Lemma 12.1, the number of vertices of degree at least two is bounded by $p_2 + 2k_2$. Every vertex of degree one in G_2 is adjacent to a vertex of larger degree, thus it makes never sense to choose it as a center (its neighbor will always be cheaper). Hence, it suffices to enumerate every set S_2 of p_2 vertices of degree larger than one and test in linear time, as per Lemma 12.3, whether a solution inside the budget k_2 is possible. Using Proposition 12.2 we can bound the running time by

$$\binom{p_2 + 2k_2}{p_2} \cdot pk + O(n + m) = O(2^{3\sqrt{p_2 k_2}} + n + m).$$

We are left with the cases where p_1 or p_2 are equal to zero: then the only possible solution is to remove *all* edges within G_1 or G_2, respectively, and connect all the resulting isolated vertices to an arbitrary center in the other instance. We either follow through with the operation, if within the respective budget, or deduce that the subinstance is not solvable. We conclude that the above algorithm will at some point guess the correct budgets for G_1 and G_2 and thus find a solution of size at most k. The theorem follows.

Theorem 13. *p-Starforest Editing is solvable in time $O(2^{3\sqrt{pk}} + n + m)$.*

12.2 Subexponential time bicluster editing

In this section we lift the result of Section 12.1 by showing that the following problem is solvable in time $2^{O(p\sqrt{k}\log(pk))} + O(n+m)$. Observe that we lose the subexponential dependence on p, however, contrary to the result of Misra et al. [MPS13], for fixed (or small, relative to k) p, this still is truly subexponential parameterized by k.

p-Bicluster Editing parameterized by p, k

Input:	A graph $G = (V, E)$ and a non-negative integer k.
Question:	Is there a set F of at most k edges such that $G \triangle F$ is a disjoint union of p complete bipartite graphs?

We denote a biclique of G as $C = (A, B)$ and call the sets A, B the *sides* of C. Before describing the algorithm for the general problem, we show that the following simpler problem is solvable in linear time using a greedy algorithm:

Annotated Bicluster Editing

Input:	A bipartite graph $G = (A, B, E)$, a partition $\mathcal{A} = \{A_1, A_2, \ldots, A_p\}$ of A and a non-negative integer k.
Question:	Is there a set F of at most k edges such that $G \triangle F$ is a disjoint union of p complete bipartite graphs with each one side in $\mathcal{A}$?

Lemma 12.4. Annotated Bicluster Editing *is solvable in time* $O(n+m)$.

Proof. Let $G = (A, B, E)$, $\mathcal{A} = \{A_1, \ldots, A_p\}$, k be an instance of Annotated Bicluster Editing. Consider a vertex $v \in B$ and define $\text{cost}_i(v)$ to be the cost of placing v in B_i where $C_i = (A_i, B_i)$ is the ith biclique of the solution, i.e.,

$$\text{cost}_i(v) = |A_i| - 2\deg_{A_i}(v) + \deg(v),$$

where $\deg_{A_i}(v) = |N(v) \cap A_i|$. We prove the following claim which implies that we can greedily assign each vertex $v \in B$ to a biclique of minimum cost.

Claim 12.5. *An optimal solution will always have $v \in B$ in a biclique $C_i = (A_i, B_i)$ which minimizes* $\text{cost}_i(v)$.

Suppose that $\text{cost}_i(v)$ is minimal but v is placed by a solution F in a biclique $C_j = (A_j, B_j)$ with $\text{cost}_j(v) > \text{cost}_i(v)$. Deleting from F all edges E_j between v and A_j and adding all edges E_i between v and A_i creates a new solution $F' = F \setminus E_j \cup E_i$. Since $\text{cost}_j(v) > \text{cost}_i(v)$, we have that $|F| > |F'|$ hence F is not optimal. This concludes the proof of the claim and the lemma. □

12.2.1 Bicluster editing

We now show that the problem p-Bicluster Editing is solvable in subexponential time by using the kernel from Theorem 5, guessing the annotated sets and applying the polynomial time algorithm for the annotated version of the problem. The important ingredient will be *cheap* vertices, by which we mean vertices that are known to receive very few edits. Intuitively, a cheap vertex is a "pin" that in subexponential time reveals for us its neighborhood in the solution, and thus can be leveraged to uncover parts of said solution.

We adopt the following notation and vocabulary. For an instance (G, k) of p-Bicluster Editing, and a solution F, we call $H = G \triangle F$ the *target* graph. A vertex v is called *cheap* with respect to F if it receives at most $\sqrt{k}$ edits. Observe that any set X of size larger than $2\sqrt{k}$ has a cheap vertex. We call such a set *large* and all sets that contain at most $2\sqrt{k}$ vertices *small*. We will further classify the bicliques in the target graph into two different classes: A biclique is small if its vertex set is small and large otherwise.

The algorithm now works as follows. Given an input instance (G, k) of p-Bicluster Editing, we try all combinations of $p_s + p_\ell = p$, with the intended meaning that p_s is the number of small bicliques and p_ℓ is the number of large bicliques in the target graph.

Handling small bicliques. We enumerate a set of p_s sets $\mathcal{A}_s \subseteq 2^V$ with the property that they are pairwise disjoint, and each of size at most $2\sqrt{k}$. Furthermore, $G[\bigcup \mathcal{A}_s]$ contains at most k edges. Delete all edges in $\mathcal{A}_s$ and reduce the budget accordingly. These are going to be all the left sides in small bicliques. This enumeration takes time

$$(2\sqrt{k})^{p_s}\binom{n}{2\sqrt{k}}^{p_s} \leq (2\sqrt{k})^{p}\binom{pk+k^2}{2\sqrt{k}}^{p} = 2^{O(p\sqrt{k}\log(pk))}.$$

Handling large bicliques. The large bicliques have the following nice property. Since the vertex set of each such biclique is large, every biclique contains a cheap vertex. We guess a set $\mathcal{B}_\ell$ of size p_ℓ. For the biclique C_i, the vertex v_i of $\mathcal{B}_\ell$ will be a cheap vertex in B_i. Now, we enumerate all combinations of p_ℓ sets $\mathcal{N} = \langle N_1, N_2, \ldots, N_{p_\ell}\rangle$, each of size at most $2\sqrt{k}$ which will be the edited neighborhood of each cheap vertex, and we conclude that $A_i = N_H(v_i) = N_G(v_i) \triangle N_i$. The enumeration of this asymptotically takes time

$$\binom{n}{p_\ell} \cdot (2\sqrt{k})^{p_\ell}\binom{n}{2\sqrt{k}}^{p_\ell} \leq \binom{pk+k^2}{p} \cdot (2\sqrt{k})^{p}\binom{pk+k^2}{2\sqrt{k}}^{p} = 2^{O(p\sqrt{k}\log(pk))}.$$

Putting things together. With the above two steps, in time $2^{O(p\sqrt{k}\log(pk))}$ we obtained all the left sides $\mathcal{A}$, partitioned into $\mathcal{A}_s$ and $\mathcal{A}_\ell$. Using this information, we can in polynomial time compute whether the Annotated Bicluster Editing instance $(G, k, \mathcal{A})$ is a yes-instance. If so, we conclude yes, otherwise, we backtrack.

Theorem 14. *p-Bicluster Editing is solvable in time $2^{O(p\sqrt{k}\log(pk))}+O(n+m)$.*

Proof. We now show that the algorithm described above correctly decides p-Bicluster Editing given an instance (G,k). Suppose that the algorithm above concludes that (G,k) is a yes-instance. The only time it outputs yes, is when Annotated Bicluster Editing for a given set $\mathcal{A}$ and a given budget k' outputs yes. Since this budget is the leftover budget from making A an independent set, it is clear that any Annotated Bicluster Editing solution of size at most k' gives a yes-instance for p-Bicluster Editing.

Suppose now for the other direction that (G,k) is a yes-instance with F a solution for p-Bicluster Editing. Consider the left sides $A_1,\ldots,A_p$ of $G \triangle F$ with the restriction that the smaller of the two sides in C_i is named A_i. First we observe that during our subexponential time enumeration of sets, all the A_is that are of size at most $2\sqrt{k}$ will be enumerated in one of the branches where p_s is set to the number of small bicliques. Furthermore, if A_j is large, then both are large, and then, for each of the large bicliques, there is a branch where we selected exactly one cheap vertex for each of the largest sides. Given these cheap vertices, there is a branch where we guess exactly the edits affecting each of the cheap vertices, hence we can conclude that in some branch, we know the entire partition $\mathcal{A}$. From Lemma 12.4, we can conclude that the algorithm described above concludes correctly that we are dealing with a yes-instance. □

12.2.2 The t-partite case

We can in fact obtain similar (we treat t here as a constant so the results are up to some constant factors in the exponents) results for the more general case of t-partite p-cluster Editing. Again we need the polynomial kernel described in Theorem 5. The only difference now to the bicluster case is that we define a cluster to be small if *every side* is small. In this case, we can enumerate $\binom{n}{\sqrt{k}}^{tp}$ sets, which will form the small clusters.

In the other case a cluster $C=(A_1,A_2,\ldots,A_t)$ is divided into $A_1,A_2,\ldots,A_{t_s}$ small sides and $A_{t_s+1},A_{t_s+2},\ldots,A_t$ large sides. For this case, we guess *all* the small sides and for each of the large sides we guess a cheap vertex. Guessing the neighborhoods $N_{t_s+1},N_{t_s+2},\ldots,N_t$ for the cheap vertices $v_{t_s+1},v_{t_s+2},\ldots,v_t$ gives us complete information on C; To compute what A_j is, if $j>t_s$, we simply take the intersection $\bigcap_{t_s<i\leq t,i\neq j} N_i$ and remove $\bigcup_{i\leq t_s} A_i$. We arrive at the following lemma whose proof is directly analogous to that of Theorem 14.

Theorem 15. *The problem t-partite p-cluster Editing is solvable in subexponential time $2^{O(p\sqrt{k}\log(pk))}+O(n+m)$.*

Part IV

Lower bounds

In this last of the technical parts we present some lower bounds results. We start with two NP-completeness results, showing that Threshold Editing and Chain Editing are NP-complete in Chapter 13. We then move on to lower bounds for subexponentiality, that is, we list some problems for which a subexponentiality result would break the *exponential time hypothesis*. These are Trivially Perfect Editing and Trivially Perfect Deletion, as well as Cograph Deletion and Cograph Editing in Chapter 14. Then we show that C_4-free Deletion nor C_4-free Completion admit subexponential time algorithms—as well as state why we study these problems—in Chapter 15.

Finally, in Chapter 16, we show that, unless we limit the number of connected components in the target graph, neither Starforest Editing nor Bicluster Editing are solvable in subexponential time, again under the exponential time hypothesis. We complement this result by showing that parameterizing on the budget is also necessary; p-Starforest Editing is W[1]-hard when parameterized by p alone.

Chapter 13

Threshold and chain graphs

In this chapter we show that Threshold Editing is NP-complete. Recalling (see Figure 2.4) that chain graphs are bipartite graphs with structure very similar to that of threshold graphs, it should not be surprising that we obtain as a corollary that Chain Editing is NP-complete as well. The computational complexity of Threshold Editing and Chain Editing has repeatedly been stated as open interesting questions, starting from Natanzon, Shamir, and Sharan [NSS01], and then more recently by Burzyn, Bonomo, and Durán [BBD06], and again very recently by Liu, Wang, Guo, and Chen [LWGC12].

We conclude the chapter by giving a proof for the fact that Chordal Editing is NP-complete.[1] The problem was recently shown to be FPT by Cao and Marx [CM14], however they studied a more general problem which indeed is well-known to be NP-complete as it is a generalized version of Chordal Vertex Deletion.

13.1 NP-completeness of Threshold Editing

In this section we show the following:

Theorem 16. Threshold Editing *is* NP*-complete, even on split graphs.*

A boolean formula φ is in 3-CNF-SAT if it is in conjunctive normal form and each clause has at most three variables. Our hardness reduction is from the problem 3Sat, where we are given a 3-CNF-SAT formula φ and asked to decide whether φ admits a satisfying assignment. We will denote by $\mathcal{C}_\varphi$ the set of clauses, and by $\mathcal{V}_\varphi$ the set of variables in a given 3-CNF-SAT formula φ. An *assignment* for a formula φ is a function $\alpha\colon \mathcal{V}_\varphi \to \{\texttt{true}, \texttt{false}\}$. Furthermore, we assume we have some natural lexicographical ordering $<_{\text{lex}}$ of the clauses $c_1, \ldots, c_{|\mathcal{C}_\varphi|}$ and the same for the variables $v_1, \ldots, v_{|\mathcal{V}_\varphi|}$, hence we may write, for some variables x and y,

[1] It has somehow been folklore that Chordal Editing is NP-complete. Natanzon [Nat99], Natanzon, Shamir, and Sharan [NSS01], Sharan [Sha02] all cite private communication with Ben-Dor 1996. I have been unable to locate a proof for the NP-completeness of Chordal Editing in the literature, and therefore choose to include the observation.

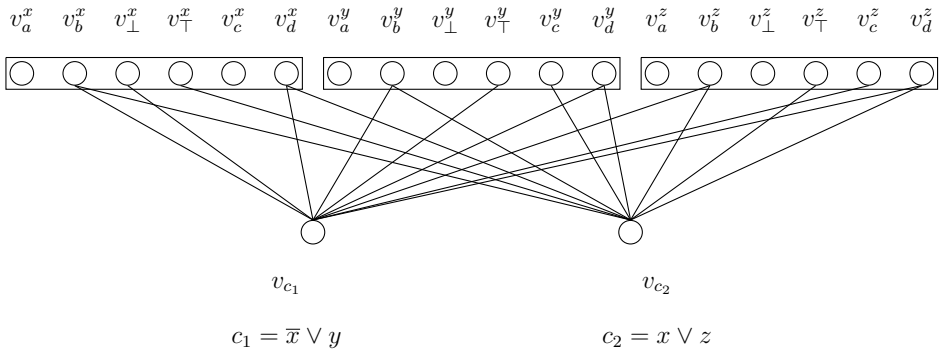


Figure 13.1: The connections between clause gadgets and variable gadgets. All the vertices on the top (the variable vertices) belong to the clique, while the vertices on the bottom (the clause vertices) belong to the independent set. The vertices in the left part of the clique has higher degree than the vertices of the right part of the clique, whereas the clause vertices (in the independent set) will all have the same degree, namely $3 \cdot |\mathcal{V}_\varphi|$.

that $x <_{\text{lex}} y$. To immediately get an impression of the reduction we aim for, the construction is depicted in Figure 13.1.

Construction. We now want to construct a graph G_φ and pick a corresponding integer k_φ so that (G_φ, k_φ) is a yes-instance of Threshold Editing if and only if φ is satisfiable. We will design G_φ to be a split graph, so that the split partition is forced to be maintained in any threshold graph within distance k_φ of G_φ, where $k_\varphi = |\mathcal{C}_\varphi| \cdot (3|\mathcal{V}_\varphi| - 1)$.

Given φ, we first create a clique of size $6|\mathcal{V}_\varphi|$. To each variable $x \in \mathcal{V}_\varphi$, we associate six vertices of this clique, and order them in the following manner

$$v_a^x, v_b^x, v_\perp^x, v_\top^x, v_c^x, v_d^x.$$

We will throughout the reduction refer to this ordering as π_φ: π_φ is a partial order which has

$$v_a^x <_{\pi_\varphi} v_b^x <_{\pi_\varphi} v_\top^x, v_\perp^x <_{\pi_\varphi} v_c^x <_{\pi_\varphi} v_d^x,$$

and for every two vertex $v_\star^x$ and $v_\star^y$ with $x <_{\text{lex}} y$, we have $v_\star^x <_{\pi_\varphi} v_\star^y$. Observe that we do not specify which comes first of $v_\top^x$ and $v_\perp^x$—this is the choice that will result in the assignment α for φ.

We enforce this ordering by adding $O(k_\varphi^2)$ vertices in the independent set; Enforcing that v_1 comes before v_2 in the ordering is done by adding $k_\varphi + 1$ vertices in the independent set incident to all the vertices coming before v_1, including v_1. Since swapping the position of v_1 and v_2 would demand at least $k_\varphi + 1$ edge modifications and k_φ is the intended budget, in any yes-instance, v_1 ends up before v_2 in the ordering of the clique.

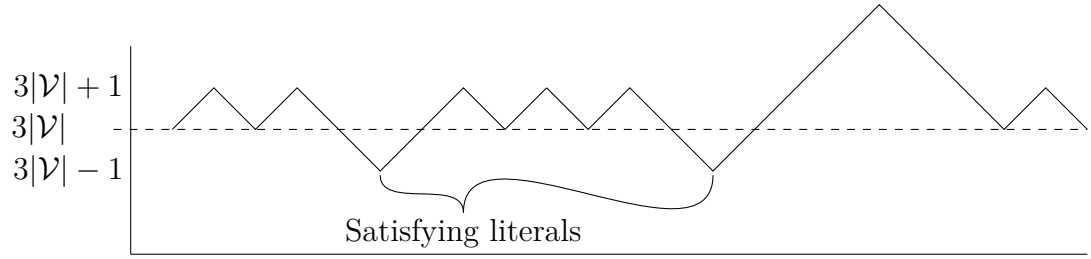


Figure 13.2: The cost with which we charge a clause vertex depends on the cut-off point; The x-axis denotes the point in the lexicographic ordering which separates the vertices adjacent to the clause vertex from the vertices not adjacent to the clause vertex.

We proceed adding the clause gadgets; For every clause $c \in \mathcal{C}_\varphi$, we add one vertex v_c to the independent set. Hence, the size of the independent set is $O(|\mathcal{C}_\varphi| + k_\varphi^2)$. For a variable x occurring in c, we add an edge between v_c and $v_\perp^x$ if it occurs negatively, and between v_c and $v_\top^x$ otherwise. In addition, we make v_c incident to v_b^x and v_d^x.

For a variable z which does not occur in a clause c, we make v_c adjacent to v_b^z, v_c^z, and v_d^z. To complete the reduction, we add $4(k_\varphi + 1)$ isolated vertices; $k_\varphi + 1$ vertices to the left in the independent set, $k_\varphi + 1$ vertices to the right in the independent set, and $k_\varphi + 1$ to the left and $k_\varphi + 1$ to the right in the clique. This ensures that no vertex will move from the clique to the independent set partition, and vice versa.

Properties of the constructed instance. Before proving Theorem 16, and specifically Lemma 13.4, we may observe the following, which may serve as an intuition for the idea of the reduction. When we consider a fixed permutation of the variable gadget vertices (the clique side), the only thing we need to determine for a clause vertex v_c, is the *cut-off point*: the point in π_φ at which the vertex v_c will no longer have any neighbors. Observing that no vertex v_i^x swaps places with any other v_j^x for $i, j \in \{a, b, c, d\}$, and that no $v_\star^x$ changes with $v_\star^y$ for $x, y \in \mathcal{V}_\varphi$, consider a fixed permutation of the variable vertices. We charge the clause vertices with the edits incident to the clause vertex. Since the budget is $k_\varphi = |\mathcal{C}| \cdot (3|\mathcal{V}_\varphi| - 1)$, and every clause needs at least $3|\mathcal{V}_\varphi| - 1$, to obtain a solution (upcoming Lemma 13.2) we need to charge every clause vertex with exactly $3|\mathcal{V}_\varphi| - 1$ edits. Figure 13.2 illustrates the charged cost of a clause vertex.

Observation 13.1. *The graph G_φ resulting from the above procedure is a split graph and when $k_\varphi = |\mathcal{C}| \cdot (3|\mathcal{V}_\varphi| - 1)$, if H is a threshold graph within distance k_φ of G_φ, H must have the same clique-maximizing split partition as G_φ.*

Lemma 13.2. *Let (G_φ, k_φ) be a yes-instance to* Threshold Editing *constructed from a 3-CNF-SAT formula φ with $|F| \leq k_\varphi$ a solution. For any clause vertex v_c, at least $3|\mathcal{V}_\varphi| - 1$ edges in F are incident to v_c.*

Proof. By the properties of π_φ, we know that the only vertices we may change the order of are those corresponding to $v^\star_\top$ and $v^\star_\bot$. Pick any index in π_φ for which we know that v_c is adjacent to all vertices on the left hand side and non-adjacent to all vertices on the right hand side. Let L_c be the set of variables whose vertices are completely adjacent to v_c and R_c the corresponding set completely non-adjacent to v_c. By construction, v_c has exactly three neighbors in each variable and thus these variable gadgets contribute $3(|L_c| + |R_c|)$ to the budget. If $L_c \cup R_c = \mathcal{V}_\varphi$, we are done, as v_c needs at least $3|\mathcal{V}_\varphi|$ edits here.

Suppose therefore that there is a variable x whose vertex v^x_a is adjacent to v_c and v^x_d is non-adjacent to v_c. But then we have already deleted the existing edge $v_c v^x_d$ and added the non-existing edge $v_c v^x_a$. This immediately gives a lower bound on $3(|\mathcal{V}_\varphi| - 1) + 2 = 3|\mathcal{V}_\varphi| - 1$ edits. □

Proof of correctness. We proceed to show that the reduction above is correct, and henceforth, when F is a set of edges and v a vertex, denote by $F(v)$ the set of edges in F that are incident with v.

Lemma 13.3. *If there is an editing set F of size at most k_φ for an instance (G_φ, k_φ) constructed from a 3-CNF-SAT formula φ, and $|F(v_c)| = 3|\mathcal{V}_\varphi| - 1$, then the $<_{\text{lex}}$-highest vertex connected to v_c corresponds to a variable satisfying the clause c.*

Proof. From the proof of Lemma 13.2, we may observe that for a clause c to be within budget, we must choose a cut-off point inside a variable gadget, meaning that there is a variable x for which v_c is adjacent to v^x_a and non-adjacent to v^x_d.

We now distinguish two cases, *(i)* x is a variable occurring (w.l.o.g. positively) in c and *(ii)* x does not occur in c. For *(i)*, v_c was adjacent to v^x_b, $v^x_\top$, and v^x_d. By assumption, we add the edge to v^x_a and delete the edge to v^x_d. But then we have already spent the entire budget, hence the only way this is a legal editing, $v^x_\top$ must come before $v^x_\bot$, and hence satisfies v_c. See Figure 13.3.

For *(ii)* we have that v_c was adjacent to v^x_b, v^x_c, and v^x_d. Here we, again by assumption, add the edge to v^x_a and delete the edge to v^x_d. This alone costs two edits, so we are done. But observe that these two edits alone are not enough, hence if we want to achieve the goal of $3|\mathcal{V}_\varphi| - 1$ edited edges, the cut-off index must be inside a variable gadget corresponding to a variable occurring in c, i.e. *(i)* must be the case. □

The above result showed how to obtain a satisfying vertex for each clause from an editing set. We will now use this to get the full correctness result, which immediately implies the NP-completeness of Threshold Editing.

Lemma 13.4. *A 3-CNF-SAT formula φ is satisfiable if and only if (G_φ, k_φ) is a yes-instance to* Threshold Editing.

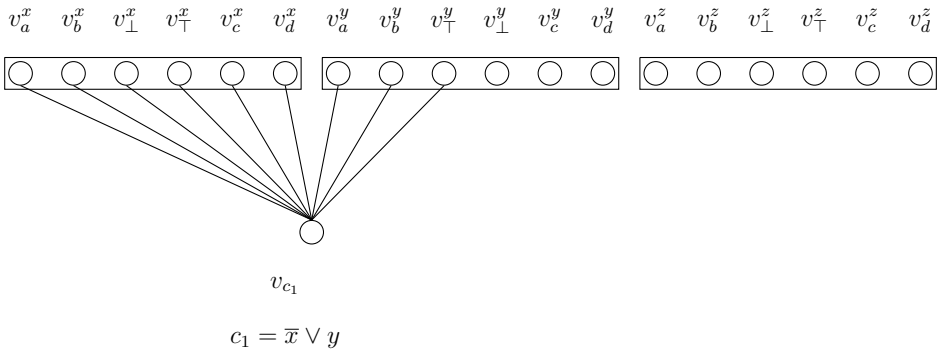


Figure 13.3: The edited version when y satisfies c_1. We have added three edges to the gadget x and deleted three edges to the gadget z, and added the edge to v_a^y and deleted the edge to v_d^y, that is, we have edited exactly $3 \cdot 2 + 2 = 3(|\mathcal{V}| - 1) + 2 = 3|\mathcal{V}| - 1$ edges incident to c_1. Notice that if $v_\perp^y$ was coming before $v_\top^y$, we would have to choose a different variable to satisfy c_1.

Proof. For the forward direction, let φ be a satisfiable 3-CNF-SAT formula where $\alpha \colon \mathcal{V}_\varphi \to \{\texttt{true}, \texttt{false}\}$ is any satisfying assignment, and (G_φ, k_φ) the Threshold Editing instance as described above. Furthermore, let π be any permutation of the vertices of the clique side with the following properties

- for every $x <_{\text{lex}} y \in \mathcal{V}_\varphi$, we have $v_\star^x <_\pi v_\star^y$,
- for every $x \in \mathcal{V}_\varphi$, we have $v_a^x <_\pi v_b^x <_\pi v_\top^x < v_c^x <_\pi v_d^x$ and $v_a^x <_\pi v_b^x <_\pi v_\perp^x < v_c^x <_\pi v_d^x$, and finally
- for every $x \in \mathcal{V}_\varphi$, we have $v_\perp^x <_\pi v_\top^x$ if and only if $\alpha(x) = \texttt{false}$.

We now show how to construct the threshold graph H_φ^π from the constructed graph G_φ by editing exactly $k_\varphi = |\mathcal{C}| \cdot (3|\mathcal{V}_\varphi| - 1)$ edges. For a clause c, let x be any variable satisfying c. If x appears positively, add every non-existing edge from v_c to every vertex $v \leq_\pi v_\top^x$ and delete all the rest. If x appears negated, use $v_\perp^x$ instead. We break the remainder of the proof in the forward direction into two claims:

Claim 13.5. *H_φ^π is a threshold graph.*

Proof of Claim 13.5. Let G_φ and π be given, both adhering to the above construction. Since G_φ is a split graph, π a total ordering of the elements in the independent set part and every vertex of the clique part of H_φ^π sees a prefix of the vertices of the independent set, their neighborhoods are naturally nested. Hence H_φ^π is a threshold graph by Proposition 2.20. ⌟

Claim 13.6. $\left|E(G_\varphi) \vartriangle E(H_\varphi^\pi)\right| = k_\varphi$.

Proof of Claim 13.6. Since we did not edit any of the edges within the clique part nor the independent set part, we only need to count the number of edits going between a clause vertex and the variable vertices. Let c be any clause and x the lexicographically smallest variable satisfying c. Suppose furthermore, without loss of generality, that x appears positively in c and has thus $\alpha(x) = \texttt{true}$. We now show that $|F(v_c)| = 3|\mathcal{V}_\varphi| - 1$, and since c was arbitrary, this concludes the proof of the claim. Since v_c is adjacent to exactly three vertices per variable, and non-adjacent to exactly three vertices per variable, we added all the edges to the vertices appearing before x and removed all the edges to the vertices appearing after x. This cost exactly $3(|\mathcal{V}_\varphi| - 1) = 3|\mathcal{V}_\varphi| - 3$, hence we have two edges left in our budget for c. Moreover, the edge $v_c v_a^x$ was added and the edge $v_c v_d^x$ was deleted. Now, c is adjacent to every vertex to the before, and including, x, and non-adjacent to all the vertices after x. The budget used was $3(|\mathcal{V}_\varphi| - 1) + 2 = 3|\mathcal{V}_\varphi| - 1$. Hence, the total number of edges edited to obtain H_φ^π is $\sum_{c \in \mathcal{C}} 3|\mathcal{V}_\varphi| - 1 = |\mathcal{C}| \cdot (3|\mathcal{V}_\varphi| - 1) = k_\varphi$. ⌟

This shows that if φ is satisfiable, then (G_φ, k_φ) is a yes-instance of Threshold Editing.

In the reverse direction, let (G_φ, k_φ) be a constructed instance from a given 3-CNF-SAT formula φ and let F be a minimal editing set such that $G_\varphi \triangle F$ is a threshold graph and $|F| \leq k_\varphi$. We aim to construct a satisfying assignment $\alpha \colon \mathcal{V}_\varphi \to \{\texttt{true}, \texttt{false}\}$ from $G_\varphi \triangle F$. By Observation 13.1, $H = G_\varphi \triangle F$ has the same split partition as G_φ. By construction, we have enforced the ordering, π_φ, of each of the vertices corresponding to the variables. Thus, we know exactly how H looks, with the exception of the internal ordering of each literal and its negation. Construct the assignment α as described above, i.e., $\alpha(x) = \texttt{false}$ if and only if $v_\perp^x <_\pi v_\top^x$.

By Lemmata 13.2 and 13.3, it follows directly that α is a satisfying assignment for φ which concludes the proof of the main lemma. □

The above lemma shows that there is a polynomial time many-one (Karp) reduction from 3Sat to Threshold Editing so we may wrap up the main theorem of this section. Lemma 13.4 implies Theorem 16, that Threshold Editing is NP-complete, even on split graphs.

For the sake of the next section, devoted to the proof of Theorem 17, we work on the annotated version of editing to threshold graphs. Recall from Chapter 9.1 that in this problem, we are given a split graph and we are asked to edit the graph to a threshold graph while respecting the split partition.

Corollary 13.7. Split Threshold Editing *is* NP*-complete.*

Proof. Split Threshold Editing is clearly in NP and that the problem is NP-hard follows immediately from combining Lemma 13.4 with Observation 13.1. □

Corollary 13.8. *Assuming ETH, neither* Threshold Editing *nor* Split Threshold Editing *are solvable in* $2^{o(\sqrt{k})} \cdot \mathrm{poly}(n)$ *time.*

Proof. First we note that for any given input instance φ of 3Sat, by the definition of the budget k_φ in the reduction above,

$$\sqrt{k}_\varphi = \sqrt{|\mathcal{C}_\varphi| \cdot (3|\mathcal{V}_\varphi| - 1)} < |\mathcal{C}_\varphi| + 3|\mathcal{V}_\varphi|.$$

This implies that an $O^\star(2^{o(\sqrt{k_\varphi})})$ algorithm for either problem would yield an algorithm solving 3Sat in time $2^{o(n+m)}$, where n and m here refer to the number of variables and clauses, respectively, of φ, the input instance of 3Sat. This is in contradiction with the exponential time hypothesis, so we conclude that the corollary is true. □

13.2 NP-hardness of Chain and Chordal Editing

13.2.1 Chain Graphs

A bipartite graph $G = (A, B, E)$ is a *chain graph* if the neighborhoods of A are nested (which necessarily implies the neighborhoods of B are nested as well). Recalling Proposition 2.31, chain graphs are closely related to threshold graphs: given a bipartite graph $G = (A, B, E)$, if one replaces A (or B) by a clique, the resulting graph is a threshold graph if and only if G was a chain graph.

It immediately follows from the above exposition that the following problem is NP-complete. This problem has also been referred to as Chain Editing in the literature (for instance in the work by Guo [Guo07]).

Bipartite Chain Editing

Input:	A bipartite graph $G = (A, B, E)$ and an integer k
Question:	Does there exist a set $F \subseteq A \times B$ of size at most k such that $G \triangle F$ is a chain graph?

Observe that in this problem we are given a bipartite graph together with a bipartition, and are asked to respect the bipartition in the editing set.

Corollary 13.9. *The problem* Bipartite Chain Editing *is* NP*-complete.*

Proof. We reduce from Split Threshold Editing. Recall that to this problem, we are given a split graph $G = (V, E)$ with split partition (C, I), and an integer k, and asked whether there is an editing set $F \subseteq C \times I$ of size at most k such that $G \triangle F$ is a threshold graph. Since a chain graph is a threshold graph with the edges in the clique C removed (Proposition 2.31) it follows that $G \triangle F$ with all the edges in the clique removed is a chain graph.

Let (G, k) be the input to Split Threshold Editing and let (C, I) be the split partition. Remove all the edges in C to obtain a bipartite graph $G' = (A, B, E')$. Now it follows directly from Proposition 2.31 that (G, k) is a yes-instance to Split Threshold Editing if and only if (G', k) is a yes-instance to Bipartite Chain Editing. □

CHAIN EDITING

Input:	A graph $G = (V, E)$ and a non-negative integer k
Question:	Is there a set F of size at most k such that $G \triangle F$ is a chain graph?

We now aim to prove the following theorem:

Theorem 17. CHAIN EDITING *is* NP*-complete.*

Proof. Reduction from BIPARTITE CHAIN EDITING. Let $G = (A, B, E)$ be a bipartite graph and consider the input instance (G, k) of BIPARTITE CHAIN EDITING. We now show that adding $2(k + 1)$ new vertices to G to obtain a graph $G' = (V, E')$, gives us that (G', k) is a yes-instance of CHAIN EDITING if and only if (G, k) is a yes-instance of BIPARTITE CHAIN EDITING.

Let $G = (A, B, E)$ be a bipartite graph and k a positive integer. Add $k + 1$ new vertices $a_1, \ldots, a_{k+1}$ to A and make them universal to B, and add $k + 1$ new vertices $b_1, \ldots, b_{k+1}$ to B and make them universal to A. Call the resulting graph $G' = (V, E')$. The following claim follows immediately from the construction.

Claim 13.10. *If $G' \triangle F$ is a chain graph with $|F| \leq k$, then $G' \triangle F$ has bipartition $(A \cup \{a_1, \ldots, a_{k+1}\}, B \cup \{b_1, \ldots, b_{k+1}\})$.*

It follows that for any input instance (G, k) of BIPARTITE CHAIN EDITING, the instance (G', k) as constructed above is a yes-instance of CHAIN EDITING if and only if (G, k) is a yes-instance of BIPARTITE CHAIN EDITING. □

Due to the similarity of CHAIN EDITING to THRESHOLD EDITING, it should not be surprising that we obtain similar lower bounds under ETH as we did in the previous section.

Corollary 13.11. *Assuming ETH, there is no algorithm solving neither* CHAIN EDITING *nor* BIPARTITE CHAIN EDITING *in time $2^{o(\sqrt{k})} \cdot \text{poly}(n)$.*

Proof. In both these cases we reduce from SPLIT THRESHOLD EDITING without changing the parameter k. Hence this follows immediately from the above exposition and from Corollary 13.8. □

13.2.2 Chordal Graphs

We will now combine our previous result on CHAIN EDITING with the following observation of Yannakakis to prove Theorem 18. Yannakakis showed [Yan81a], while proving the NP-completeness of CHORDAL COMPLETION (more often known as MINIMUM FILL-IN), that a bipartite graph can be transformed into a chain graph by adding at most k edges if and only if the cobipartite graph formed by completing the two sides can be transformed into a chordal graph by adding at most k edges.

CHORDAL EDITING

Input: A graph G and an integer k
Question: Does there exist a set F of at most k edges such that $G \triangle F$ is chordal?

Theorem 18. CHORDAL EDITING *is* NP*-complete.*

To prove the theorem, we will first give an intermediate problem that makes the proof simpler. Let $G = (A, B, E)$ be a cobipartite graph. Define the problem COBIPARTITE CHORDAL EDITING to be the problem which on input (G, k) asks if we can edit at most k edges between A and B, i.e., does there exist an editing set $F \subseteq A \times B$ of size at most k, such that $G \triangle F$ is a chordal graph. That is, COBIPARTITE CHORDAL EDITING asks for the bipartition A, B to be respected.

COBIPARTITE CHORDAL EDITING

Input: A cobipartite graph $G = (A, B, E)$ and an integer k
Question: Does there exist a set $F \subseteq A \times B$ of size at most k such that $G \triangle F$ is a chordal graph?

We will use the following observation to prove the above theorem:

Lemma 13.12. *If $G = (A, B, E)$ is a bipartite graph, and $G' = (A, B, E')$ is the cobipartite graph constructed from G by completing A and B, then F is an optimal edge editing set for* BIPARTITE CHAIN EDITING *on input (G, k) if and only if F is an optimal edge editing set for* COBIPARTITE CHORDAL EDITING *on input (G', k).*

Proof. Let F be an optimal editing set for BIPARTITE CHAIN EDITING on input (G, k) and suppose that $G' \triangle F$ has an induced cycle of length at least four. Since G' is cobipartite, it has a cycle of length exactly four. Let $\langle a_1, b_1, b_2, a_2 \rangle$ be this cycle. But then it is clear that a_1b_1, a_2b_2 forms an induced $2K_2$ in $G \triangle F$, contradicting the assumption that F was an editing set.

For the reverse direction, suppose F is an optimal edge editing set for COBIPARTITE CHORDAL EDITING on input (G', k) only editing edges between A and B. Suppose for the sake of a contradiction that $G \triangle F$ was not a chain graph. Since F only goes between A and B, $G \triangle F$ is bipartite and hence by the assumption must have an induced $2K_2$. This obstruction must be of the form a_1b_1, a_2b_2, but then $\langle a_1, b_1, b_2, a_2 \rangle$ is an induced C_4 in $G' \triangle F$ which is a contradiction to the assumption that $G' \triangle F$ was chordal. Hence $G \triangle F$ is a chain graph. □

Corollary 13.13. COBIPARTITE CHORDAL EDITING *is* NP*-complete.*

We are now ready to prove Theorem 18, that CHORDAL EDITING is NP-complete.

Proof of Theorem 18. Let $G = (A, B, E)$ be a cobipartite graph and (G, k) the input to Cobipartite Chordal Editing. Our reduction is as follows. Create $G' = (A' \cup B', E')$ as follows:

- $A' = A \cup \{a_1, a_2, \ldots, a_{k+1}\}$,
- $B' = B \cup \{b_1, b_2, \ldots, b_{k+1}\}$,
- $E' = E \cup \bigcup_{i \leq k+1, b \in B'} \{a_i b\} \cup \bigcup_{i,j \leq k+1} \{a_i a_j, b_i b_j\}$

Finally, we create G'' as follows. For every edge $a_i a_j$ create $k+1$ new vertices adjacent to only a_i and a_j, and for every edge $b_i b_j$ create $k+1$ new vertices adjacent to only b_i and b_j This forces none of the edges in A' and in B' to be removed.

Claim 13.14. *The instance of* Chordal Editing (G'', k) *is equivalent to the instance* (G, k) *to* Cobipartite Chordal Editing.

Proof of claim. The proof of the above claim is straight-forward. If we delete an edge within A (resp. B), we create at least $k+1$ cycles of length 4, each of which uses at least one edge to delete, hence in any yes-instance, we do not edit edges within A (resp. B). Furthermore, every chordal graph remains chordal when adding a simplicial vertex, which is exactly what the $k+1$ new vertices are. ⌟

From the claim it follows that (G'', k) is a yes-instance to Chordal Editing if and only if (G, k) is a yes-instance to Cobipartite Chordal Editing. The theorem follows immediately from Corollary 13.13. □

Also here we get lower bounds similar to what we got from Threshold Editing:

Corollary 13.15. *Assuming ETH, there is no algorithm solving* Chordal Editing *in time* $2^{o(\sqrt{k})} \cdot \text{poly}(n)$.

Chapter 14

Trivially perfect graphs and cographs

In this section we show that Trivially Perfect Editing is NP-hard, and furthermore not solvable in subexponential parameterized time unless the exponential time hypothesis fails. The NP-hardness of the problem was established by Nastos and Gao [NG13] independently. However, their reduction [NG13, Theorem 3.3] starts with an instance of Exact 3-Cover with universe of size n and a set family of size m, and constructs an instance (G, k) of Trivially Perfect Editing with $k = \Theta(mn^2)$. Thus, the parameter blow-up is at least cubic, and the reduction cannot be used to establish the non-existence of a subexponential parameterized algorithm under ETH.

Theorem 19. Trivially Perfect Editing *is* NP*-complete and, under ETH, cannot be solved in time* $2^{o(k)}\,\mathrm{poly}(n)$ *nor* $2^{o(n+m)}$*, even on graphs with maximum degree* 4*.*

Here, we give a direct, linear reduction from 3Sat to Trivially Perfect Editing. Furthermore, the resulting graph in our reduction has maximum degree four. Thus, we in fact prove that even on input graphs of maximum degree four, Trivially Perfect Editing remains NP-hard, and that it does not admit a subexponential parameterized algorithm unless ETH fails. Formally, the following lemma will be proved, where, as in the previous chapter, for an input formula φ of 3Sat, by $\mathcal{V}_\varphi$ and $\mathcal{C}_\varphi$ we denote the variable and clause sets of φ, respectively:

Lemma 14.1. *There exists a polynomial-time reduction that, given an instance* φ *of* 3Sat*, returns an instance* (G_φ, k_φ) *of* Trivially Perfect Editing *such that* φ *is satisfiable if and only if* (G_φ, k_φ) *is a yes-instance. Furthermore,* $|V(G_\varphi)| = 13|\mathcal{C}_\varphi|$*,* $|E(G_\varphi)| = 18|\mathcal{C}_\varphi|$*,* $k_\varphi = 5|\mathcal{C}_\varphi|$*, and* $\Delta(G_\varphi) = 4$*.*

Lemma 14.1 clearly implies Theorem 19, and its conclusion follows from the reduction by an application of Proposition 2.41. Hence, we are left with constructing the reduction, to which the rest of this section is devoted. Our approach is similar to the technique used by Komusiewicz and Uhlmann to show the hardness of a

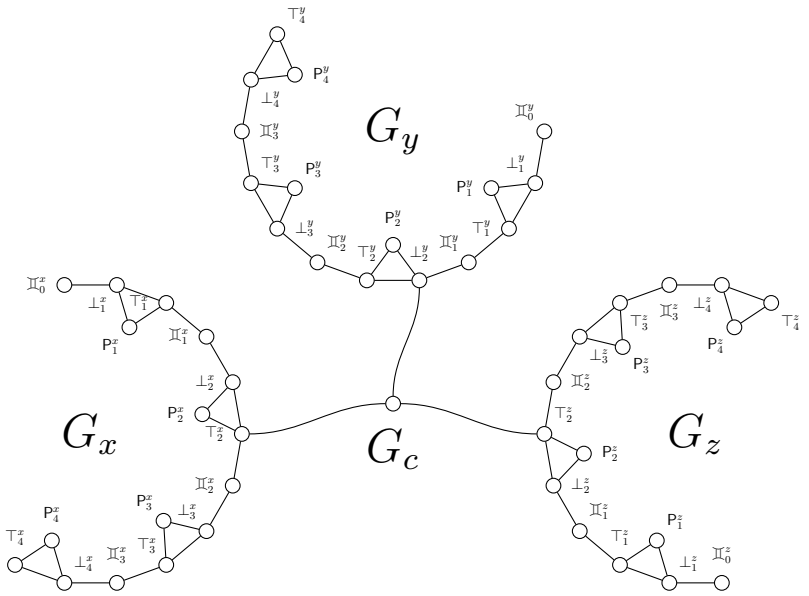


Figure 14.1: Gadget $c = x \vee \neg y \vee z$. The clause c is now the second clause all variables x, y, and z appear in, and x and z appears positively whereas y appears negatively.

similar problem, Cluster Editing [KU12]; However, the gadgets are tailored to work for the Trivially Perfect Editing problem.

We will in the very end prove that the reduction given below is already alone sufficient to show that none of the edge modification problems towards cographs are solvable in subexponential time either, unless the exponential time hypothesis fails.

14.1 Trivially perfect graphs

Let φ be the input instance of 3Sat. By standard modifications of the formula we may assume that every clause contains exactly three literals, all containing different variables, and that every variable appears in at least two clauses, and we again assume some lexicographical ordering. For a variable $x \in \mathcal{V}_\varphi$, let $p_x > 1$ be the number of occurrences of x in the clauses of φ. Observe that $\sum_{x \in \mathcal{V}_\varphi} p_x = 3|\mathcal{C}_\varphi|$. Now, for every $x \in \mathcal{V}_\varphi$ we create a variable gadget, and for every $c \in \mathcal{C}_\varphi$ we create a clause gadget.

Variable gadget. For $x \in \mathcal{V}_\varphi$, construct a graph G_x isomorphic to C_{3p_x}, a cycle on $3p_x$ vertices. The vertices of G_x are labeled $\bot_i^x, \top_i^x, \text{𝕀}_i^x$ for $i \in [0, p_x - 1]$, in the order of their appearance on the cycle. We then add a vertex P_i^x adjacent to $\top_i^x$ and $\bot_i^x$, for each $i \in [0, p_x - 1]$, see Figure 14.1. Formally, the vertices P_i^x do not belong to G_x, but they will be used to wire variable gadgets with clause gadgets. This concludes the construction of the variable gadget, and it should be clear that

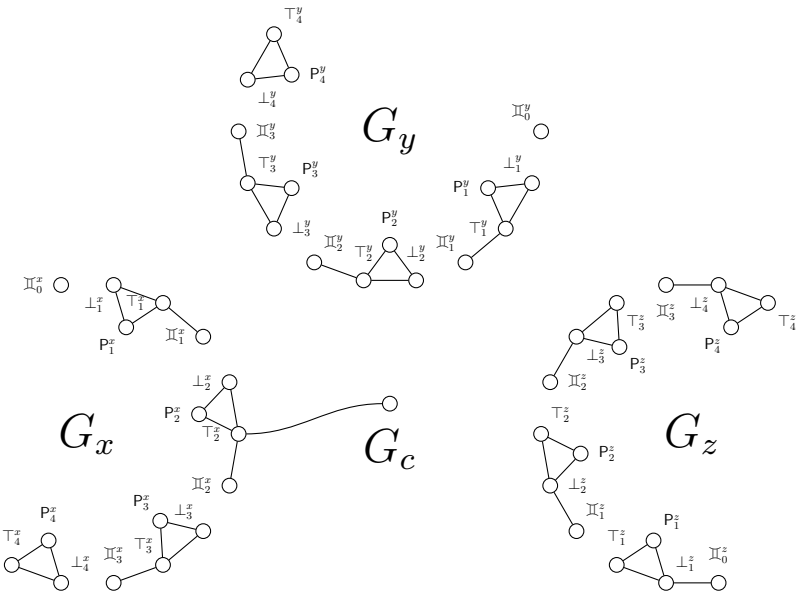


Figure 14.2: Edited gadget of $c = x \vee \neg y \vee z$ where $\alpha(x) = \texttt{true}$, $\alpha(y) = \texttt{true}$ and $\alpha(z) = \texttt{false}$ and x has been chosen (no choice, really) to satisfy c. Notice the formation of *paws*, except the one incident to c which induces a *cricket*.

the number of created vertices and edges is bounded linearly in p_x—more precisely, we created $4p_x$ vertices and $5p_x$ edges.

For the sake of later argumentation, we now define the deletion set F_x^α for G_x. If, in an assignment of variables $\alpha \colon \mathcal{V}_\varphi \to \{\texttt{false}, \texttt{true}\}$, we have $\alpha(x) = \texttt{true}$, then we let F_x^α be the set consisting of every edge of the form $\text{Ш}_i^x \bot_{i+1 \bmod p_x}^x$ for $i \in [0, p_x - 1]$. If, on the other hand, $\alpha(x) = \texttt{false}$, we define the deletion set F_x^α to be the set comprising the edges $\top_i^x \text{Ш}_i^x$ for $i \in [0, p_x - 1]$, see Figure 14.2. We will later show that these are the only relevant editing sets of size at most p_x for G_x.

Clause gadget. The clause gadgets are very simple. A clause gadget consists simply of one vertex, i.e., for a clause $c \in \mathcal{C}_\varphi$ construct the vertex v_c. This vertex will be connected to G_x, G_y and G_z, for x, y, and z being the variables appearing in c, in appropriate places, depending on whether the variable occurs positively or negatively in c. More precisely, if c is the ith clause x appears in, then we make v_c adjacent to $\top_i^x$ provided that x appears positively in c, and to $\bot_i^x$ provided that x appears negatively in c. This concludes the construction of a clause gadget. As every clause gadget contains one vertex and three edges, the construction of all the clause gadgets creates $|\mathcal{C}_\varphi|$ vertices and $3|\mathcal{C}_\varphi|$ edges.

The deletion set F_c^α for a clause gadget and a satisfying assignment $\alpha \colon \mathcal{V}_\varphi \to \{\texttt{false}, \texttt{true}\}$ will be as follows. Suppose $c = \ell_x \vee \ell_y \vee \ell_z$, where the literals ℓ_x, ℓ_y, and ℓ_z contain variables x, y, and z, respectively. Pick the lexicographically first literal satisfying c, say ℓ_x, and delete the two other edges in the connection, i.e., the two edges connecting v_c with vertices of G_y and G_z. Thus v_c remains a vertex

of degree 1, adjacent to a vertex of G_x.

Let G_φ be the constructed graph. We set the budget for edits to

$$k_\varphi = \sum_{x \in \mathcal{V}_\varphi} p_x + 2|\mathcal{C}_\varphi| = 5|\mathcal{C}_\varphi|.$$

Observe also that

$$|V(G_\varphi)| = \sum_{x \in \mathcal{V}_\varphi} 4p_x + |\mathcal{C}_\varphi| = 13|\mathcal{C}_\varphi|,$$
$$|E(G_\varphi)| = \sum_{x \in \mathcal{V}_\varphi} 5p_x + 3|\mathcal{C}_\varphi| = 18|\mathcal{C}_\varphi|,$$

and that $\Delta(G_\varphi) = 4$. Thus, all the technical properties stated in Lemma 14.1 are satisfied, and we are left with proving that (G_φ, k_φ) is a yes-instance of Trivially Perfect Editing if and only if φ is satisfiable.

Before we state the main lemma, we give two auxiliary observations that settle the tightness of the budget:

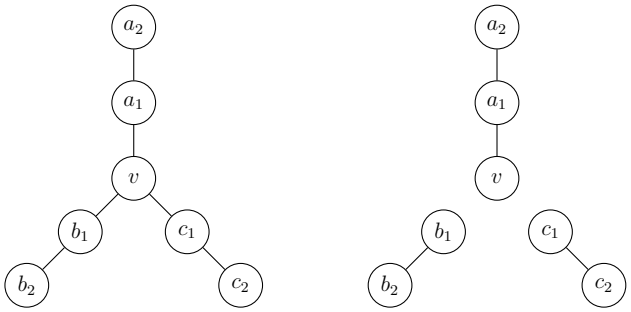
Claim 14.2. *Suppose that a graph H is a cycle on $3p$ vertices for some $p > 1$, and suppose F is an editing set for H. Then $|F| \geq p$. Moreover, if $|F| = p$ then F consists of deletions of every third edge of the cycle.*

Claim 14.3. *Suppose a graph H is a* subdivided claw*, i.e., the star $K_{1,3}$ with every leg subdivided once (see Figure 14.3a). Furthermore, suppose that F is an editing set for H. Then $|F| \geq 2$. Moreover, if $|F| = 2$ then F consists of deletions of two edges incident to the center of the subdivided claw (see Figure 14.3b).*

We will prove the two claims in order now. The astute reader should already see that this implies the tightness of the budget: every editing set needs to include exactly p_x edges of every variable gadget G_x (by Claim 14.2), and exactly two edges incident to every vertex v_c (by Claim 14.3). The additional vertices P_i^x will form the degree-1 vertices of subdivided claws created by clause gadgets, and all the subgraphs in question pairwise share at most single vertices, which means that any edit can influence at most one of them. This statement is made formal in the proof of Lemma 14.4.

Proof of Claim 14.2. Let $\langle v_0, v_1, \ldots, v_{3p-1} \rangle$ be the vertices of H, in their order of appearance on the cycle. For $i = 0, 1, \ldots, p-1$, let $A_i = \langle v_{3i}, v_{3i+1}, v_{3i+2}, v_{3i+3} \rangle$; Here and in the sequel, the indices are taken cyclically in a natural manner. Observe that each A_i induces a P_4 in H, hence $F \cap [A_i]^2 \neq \emptyset$. However, the sets $[A_i]^2$ are pairwise disjoint for $i = 0, 1, \ldots, p-1$, from which it follows that $|F| \geq p$.

Suppose now that $|F| = p$. Hence $|F \cap [A_i]^2| = 1$ for each $i \in [0, p-1]$, and there are no edits outside the sets $[A_i]^2$. There are five possible ways for an A_i of how $F \cap [A_i]^2$ can look like: It is either a deletion of the edge $v_{3i}v_{3i+1}$, $v_{3i+1}v_{3i+2}$, or $v_{3i+2}v_{3i+3}$ (henceforth referred to as types D^-, D^0, and D^+, respectively), or an addition of the edge $v_{3i}v_{3i+2}$ or $v_{3i+1}v_{3i+3}$ (henceforth called types C^- and C^+, respectively)—the sixth possibility, which has been left out, creates an induced C_4. Observe now that if some A_i has type D^-, then A_{i+1} also has type D^-, or otherwise a P_4 $\langle v_{3i+1}, v_{3i+2}, v_{3i+3}, v_{3i+4} \rangle$ would remain in the graph. Similarly, if A_i



(a) A subdivided claw.

(b) An optimally edited subdivided claw.

Figure 14.3: Vertices in and near clause gadget

has type D^+ then A_{i-1} also has type D^+. Hence, if type D^+ or D^- appears for any A_i, then all the A_is have the same type. Observe now that if some A_i had type C^- and C^+, then A_{i-1} would have to have type D^+ and A_{i+1} would have to have type D^- or otherwise an unresolved P_4 would appear; This is a contradiction with the previous observations, since types D^- and D^+ cannot appear simultaneously. Hence, we are left with only three possibilities: all the A_is have type D^-, or all have type D^0, or all have type D^+. ⌟

Proof of Claim 14.3. Denote the vertices of H as in Figure 14.3a. Consider the following three P_4s in H:

- $\langle a_2, a_1, v, c_1\rangle$,
- $\langle b_2, b_1, v, a_1\rangle$, and
- $\langle c_2, c_1, v, b_1\rangle$.

Observe that any edge addition in H can destroy at most one of these P_4s, and a deletion of any of edges a_1a_2, b_1b_2, or c_1c_2 also can destroy at most one of these P_4s. Moreover, a deletion of any of the edges incident to the center v destroys only two of them. We infer that $|F| \geq 2$ since no single edit can destroy all three considered P_4s, and moreover if $|F| = 2$, then F contains at least one deletion of an edge incident to v, say va_1. After deleting this edge we are left with a P_5 $\langle b_2, b_1, v, c_1, c_2\rangle$, and it can be readily checked that the only way to edit it to a trivially perfect graph using only one edit is to delete vb_1 or vc_1. Thus, any editing set F with $|F| = 2$ in fact consists of deletions of two edges incident to v. ⌟

Lemma 14.4. *A 3-CNF-SAT formula φ is satisfiable if and only if (G_φ, k_φ) is a yes-instance of* Trivially Perfect Editing.

Proof. Suppose φ is satisfiable and let $\alpha\colon \mathcal{V}_\varphi \to \{\texttt{false}, \texttt{true}\}$ be a satisfying assignment. Define the editing set $F^\alpha = \bigcup_{x\in\mathcal{V}_\varphi} F^\alpha_x \cup \bigcup_{c\in\mathcal{C}_\varphi} F^\alpha_c$; Note that F consists of deletions only. Then we have that $|F^\alpha| = k_\varphi$ and it can be easily seen

that $G \triangle F$ is a disjoint union of components of constant size, each being a paw or a cricket (see Table 2.1 in Section 2.2.5, or simply Figure 14.2). Both these graphs are trivially perfect, so a disjoint union of any number of their copies is also a trivially perfect graph. Thus F^α is a solution to the instance (G_φ, k_φ).

For the reverse direction, let $F \subseteq [V(G_\varphi)]^2$ be an editing set such that $G_\varphi \triangle F$ is trivially perfect, and $|F| \leq k_\varphi$. For every $x \in \mathcal{V}_\varphi$ consider the subgraph G_x. For every $c \in \mathcal{C}_\varphi$ consider the subgraph G_c induced in G by

- the vertex v_c;
- the three neighbors of v_c, say $\square^x_{i_x}$, $\square^y_{i_y}$, and $\square^z_{i_z}$, where x, y, z are variables appearing in c and each symbol $\square$ is replaced by $\bot$ or $\top$ depending whether the variable's occurrence is positive or negative; and
- the vertices $\mathsf{P}^x_{i_x}$, $\mathsf{P}^y_{i_y}$, and $\mathsf{P}^z_{i_z}$.

Observe that each G_x is isomorphic to a cycle on $3p_x$ vertices and each G_c is isomorphic to a subdivided claw. Moreover, all these subgraphs pairwise share at most one vertex, which means that sets $[V(G_x)]^2$ for $x \in \mathcal{V}_\varphi$ and $[V(G_c)]^2$ for $c \in \mathcal{C}_\varphi$ are pairwise disjoint. By Claim 14.2 we infer that $|F \cap [V(G_x)]^2| \geq p_x$ for each $x \in \mathcal{V}_\varphi$, and by Claim 14.3 we infer that $|F \cap [V(G_c)]^2| \geq 2$ for each $c \in \mathcal{C}_\varphi$. Thus

$$|F| \geq \sum_{x \in \mathcal{V}_\varphi} p_x + 2|\mathcal{C}_\varphi| = k_\varphi.$$

Hence, in fact $|F| = k_\varphi$ and all the used inequalities are in fact equalities: $|F \cap [V(G_x)]^2| = p_x$ for each $x \in \mathcal{V}_\varphi$ and $|F \cap [V(G_c)]^2| = 2$ for each $c \in \mathcal{C}_\varphi$. Using Claims 14.2 and 14.3 again, we infer that F has the following form: it consists of deletions only, from every cycle G_x it deletes every third edge, and for every vertex v_c it deletes two out of three edges incident to it. In particular, no edit is incident to any of the vertices P^x_i for $x \in \mathcal{V}_\varphi$ and $i \in [0, p_x - 1]$.

Consider now the cycle G_x; We already know that the solution deletes either all the edges $\bot^x_i \top^x_i$ for $i \in [0, p_x - 1]$, or all the edges $\top^x_i \sqcup^x_i$ for $i \in [0, p_x - 1]$, or all the edges $\sqcup^x_i \bot^x_{i+1 \bmod p_x}$ for $i \in [0, p_x - 1]$. Observe that the first case cannot happen, since then we would have an induced P_4 $\langle \bot^x_i, \mathsf{P}^x_i, \top^x_i, \sqcup^x_i \rangle$ remaining in the graph—no other edit can destroy it. Hence, one of the latter two cases happen. Construct an assignment $\alpha \colon \mathcal{V}_\varphi \to \{\texttt{false}, \texttt{true}\}$ by, for each $x \in \mathcal{V}_\varphi$, putting $\alpha(x) = \texttt{false}$ if all the edges $\top^x_i \sqcup^x_i$ are included in F, and $\alpha(x) = \texttt{true}$ if all the edges $\sqcup^x_i \bot^x_{i+1 \bmod p_x}$ are included in F. We now claim that α satisfies φ.

For the sake of contradiction, suppose that a clause $c = \ell_x \vee \ell_y \vee \ell_z$ is not satisfied by α. Let e be the edge incident to v_c which has not been removed and suppose without loss of generality that this edge connects v_c with G_x. Suppose further that $\ell_x = x$, i.e., x appears positively in c, so $e = v_c \top^x_i$ for some $i \in [0, p_x - 1]$. Since x does not satisfy c, $\alpha(x) = \texttt{false}$ and both edges $\sqcup^x_{i-1 \bmod p_x} \bot^x_i$ and $\bot^x_i \top^x_i$ are not deleted in F—the deleted edge is $\top^x_i \sqcup^x_i$. But then we have the following induced P_4: $\langle v_c, \top^x_i, \bot^x_i, \sqcup^x_{i-1 \bmod p_x} \rangle$, which contradicts the assumption

that $G_\varphi \triangle F$ is trivially perfect. The case when $\ell_x = \neg x$, i.e., x appears negatively in c, is symmetric.

Hence α is indeed a satisfying assignment for φ and we are done. □

Lemma 14.4 guarantees that the reduction is correct, and hence Theorem 19 follows by a straightforward application of Proposition 2.41. We can also observe that this reduction works immediately for Trivially Perfect Deletion as well since every optimal edit set consisted purely of deletions (see Claims 14.2 and 14.3).

Corollary 14.5. Trivially Perfect Deletion *is* NP*-complete and, under ETH, cannot be solved in time* $2^{o(k)}\,\mathrm{poly}(n)$ *nor* $2^{o(n+m)}$*, even on graphs with maximum degree* 4*.*

It might be interesting to note here that for p-Trivially Perfect Editing, editing to a graph with at most p connected components is not likely to be of help when it comes to subexponential parameterized time algorithms, as adding a universal vertex to an input graph will have an optimal solution which is connected. Recall that bounding the number of connected components in the target graph is exactly what was done to obtain subexponential time algorithms for Cluster Editing. It could be interesting to see if the above reduction immediately prove that for the graph class *threshold cluster*, the class of graph whose connected components is threshold graphs, editing and deleting towards that graph class have exactly the same properties as Trivially Perfect Editing and Trivially Perfect Deletion.

14.2 Cographs

Let us recall that since $\overline{P_4} = P_4$, the problems Cograph Deletion and Cograph Completion are polynomial time equivalent (Fact 2.7). We aim to show the following theorem, whose proof is a short modification of the proof of the above theorem, Theorem 19, and the corresponding corollary, Corollary 14.5. The NP-hardness of Cograph Editing was first shown by Liu, Wang, Guo, and Chen [LWGC12], however, their reduction from Exact 3-Cover, adapted from the proof of the NP-hardness of Cograph Deletion by El-Mallah and Colbourn [EC88] suffers a quadratic blow-up in the parameter, and has $\Omega(|C|^6)$ vertices, where $|C|$ is the number of sets in the input instance. Hence, this reduction is unsuitable for showing the kind of lower bounds we are after.

Theorem 20. Cograph Completion*,* Cograph Deletion*, and* Cograph Editing *are* NP*-complete and, under ETH, cannot be solved in time* $2^{o(k)}\,\mathrm{poly}(n)$ *nor* $2^{o(n+m)}$*, even on graphs with maximum degree* 4*.*

We first show that the theorem is true for the completion and deletion variants. This follows immediately from the following observation combined with Corollary 14.5.

Observation 14.6. *Given an instance φ of* 3Sat*, the constructed instance (G_φ, k_φ) of* Trivially Perfect Editing *in the previous section is C_4-free.*

What remains to show is that even if we allow adding edges, and possibly even constructing four-cycles, we could just as well have deleted edges and created a trivially perfect graph within the same budget. The following lemma encapsulates this, stating that Trivially Perfect Editing and Cograph Editing are equivalent on the graph class generated by the above reduction from 3Sat.

Lemma 14.7. *Given an instance φ of* 3Sat*, (G_φ, k_φ) is a yes-instance of* Trivially Perfect Editing *if and only if it is a yes-instance of* Cograph Editing *if and only if it is a yes-instance of* Cograph Deletion.

Proof. We will show that any cograph editing set of size at most k_φ can be replaced by a cograph deletion set of size at most k_φ. Hence, we show only that Trivially Perfect Editing is equivalent to Cograph Editing for the given instance. The final result will follow immediately. Since trivially perfect graphs are cographs, the forward direction is correct. We therefore focus on the reverse direction, and assume henceforth that F is a solution for (G_φ, k_φ) of Cograph Editing. We will now show that there is a solution F' of Trivially Perfect Editing. There are two things we need to take care of. First, we simply observe that the proof of Claim 14.2 immediately shows that for a cycle C_{3p}, we need at least p edits, and that there exists, for each G_x, a solution of size p_x, where p_x is the number of clauses in which x occurs. This was shown in the proof by considering all the paths on four vertices of form $A_i = \langle v_{3i}, v_{3i+1}, v_{3i+2}, v_{3i+3}\rangle$, and observing that the sets $[A_i]^2$ are pairwise disjoint, hence one edit is necessary per such occurrence.

If we can show that two edits are necessary per clause gadget, we are done, that is, we need an equivalent of Claim 14.3 (see also Figure 14.3). It can be manually verified that for the subdivided claw, however we add an edge, we need at least two more edits, which means that for the subdivided claw, we need two deletions. What remains is now to show that the clause gadget will indeed live in a subdivided claw. But since the cycle C_{3p} needs p edits, clearly in G_x, we cannot afford to delete any of the edges in the appended triangle. Hence the clause gadget will live in a subdivided claw after optimally editing all variable gadgets.

We showed that whenever there is a solution, there is a solution only deleting edges. Hence, since G_φ does not contain any four-cycles, all of the problems Trivially Perfect Editing, Trivially Perfect Deletion, Cograph Editing, and Cograph Deletion are equivalent on input (G_φ, k_φ). □

Combining Lemma 14.7 above with Lemma 14.4, the theorem of this section follows. Any subexponential parameterized time algorithm solving any of the three modification problems towards cographs would break the exponential time hypothesis.

Chapter 15

C_4-free graphs

For every $\mathcal{H}$-free Completion problem that so far turned out to be solvable in subexponential time, we had the graph C_4 in $\mathcal{H}$ together with some other graphs: trivially perfect graphs are the class excluding C_4 and P_4, threshold graphs are the class excluding $2K_2$, C_4 and P_4, and pseudosplit graphs are the class excluding $2K_2$ and C_4. Previous known subexponentiality results in the area of graph modifications are: completing to chordal graphs and chain graphs [FV13], completing to split graphs [GKK+15] and recently, completing to interval graphs [BFPP16] and proper interval graphs [BFPP14]. All these graph classes are C_4-free[1].

It is therefore natural to ask whether the C_4 is the "reason" for the existence of subexponential algorithms. However, in this chapter we show that excluding C_4 alone is not sufficient for achieving a subexponential time algorithm.

15.1 C_4-free deletion

In this section we show that C_4-free Deletion, or $2K_2$-free Completion, is NP-complete and not solvable in subexponential time. We will refer to an induced C_4 as a 4-*cycle*, or sometimes simply a *four-cycle*.

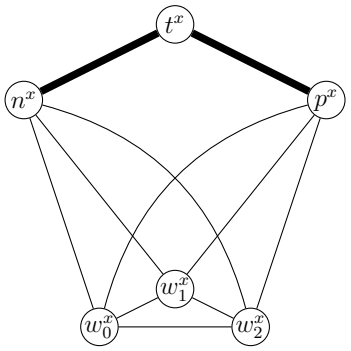
C_4-free Deletion parameterized by k

Input:	A graph G and an integer k
Question:	Does there exist a set F of at most k edges such that $G - F$ is C_4-free?

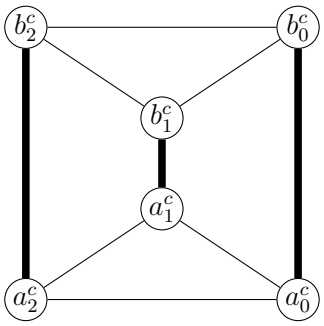
We show the following theorem.

Theorem 21. *C_4-free Deletion, or $2K_2$-free Completion is NP-complete and, under ETH, cannot be solved in time $2^{o(k)}\,\mathrm{poly}(n)$ nor $2^{o(n+m)}$.*

[1]Even p-Cluster Editing and p-Starforest Editing are C_4-free. The only problem even resembling an exception is p-Bicluster Editing; C_4 is a biclique. However, this problem is not an $\mathcal{H}$-free Editing problem for any $\mathcal{H}$ since p-bicluster graphs are not hereditary.



(a) The variable gadget G_x for a variable x with three occurrences of C_4. The edge $t^x p^x$ is the true edge and the edge $t^x n^x$ is the false edge. All C_4s of G_x can be eliminated by removing the true or the false edge.



(b) The clause gadget G_c for a clause c has three occurrences of C_4, which can be eliminated by removing two of the variable-edges, the thick edges in the figure.

Figure 15.1: Variable (left) and clause (right) gadgets used in the reduction.

Of the two equivalent problems, deletion towards C_4-free graphs and completion towards $2K_2$-free graphs, we find it much more convenient to show the hardness of C_4-FREE DELETION. Dealing with $2K_2$-free graphs is rather painful.

Construction. We reduce from 3SAT using the gadgets in Figure 15.1. Let φ be an instance of 3SAT. We construct the instance (G_φ, k_φ) for C_4-FREE DELETION and we begin by defining the graph G_φ. For every variable $x \in \mathcal{V}_\varphi$, we construct a variable gadget graph G_x. The graph G_x consists of six vertices w_0^x, w_1^x, w_2^x, n^x (for *negative*), p^x (for *positive*), and t^x. The three vertices w_0^x, w_1^x and w_2^x will induce a triangle whereas n^x and p^x are adjacent to the vertices in the triangle and to t^x. We can observe that the four vertices n^x, t^x, p^x, w_i^x induce a C_4 for $i = 0, 1, 2$, and that no other induced C_4 occurs in the gadget (see Figure 15.1a). It can also be observed that by removing either one of the edges $n^x t^x$ and $p^x t^x$, the gadget becomes C_4-free. We will refer to the edge $t^x p^x$ as the *true edge* and to $t^x n^x$ as the *false edge*. These edges are the thick edges in Figure 15.1a. This concludes the variable gadget construction.

For every clause $c \in \mathcal{C}_\varphi$, we construct a clause gadget graph G_c as follows. The graph G_c consists of two triangles, a_0^c, a_1^c, a_2^c and b_0^c, b_1^c, b_2^c. We also add the edges $a_0^c b_0^c$, $a_1^c b_1^c$, and $a_2^c b_2^c$. These three latter edges will correspond to the variables contained in c and we refer to them as *variable-edges* (the thick edges in Figure 15.1b). No more edges are added. The clause gadget can be seen in Figure 15.1b. Observe that there are exactly three induced C_4s in G_c, all of the form $a_i^c, a_{i+1}^c, b_{i+1}^c, b_i^c$ for $i = 0, 1, 2$, where the indices are taken modulo 3. Moreover, removing any two edges of the form $a_i^c b_i^c$ for $i = 0, 1, 2$ eliminates all the induced C_4s contained in G_c.

To conclude the construction, we give the connections between variable gadgets and clause gadgets that encode literals in the clauses (see Figure 15.2). If a

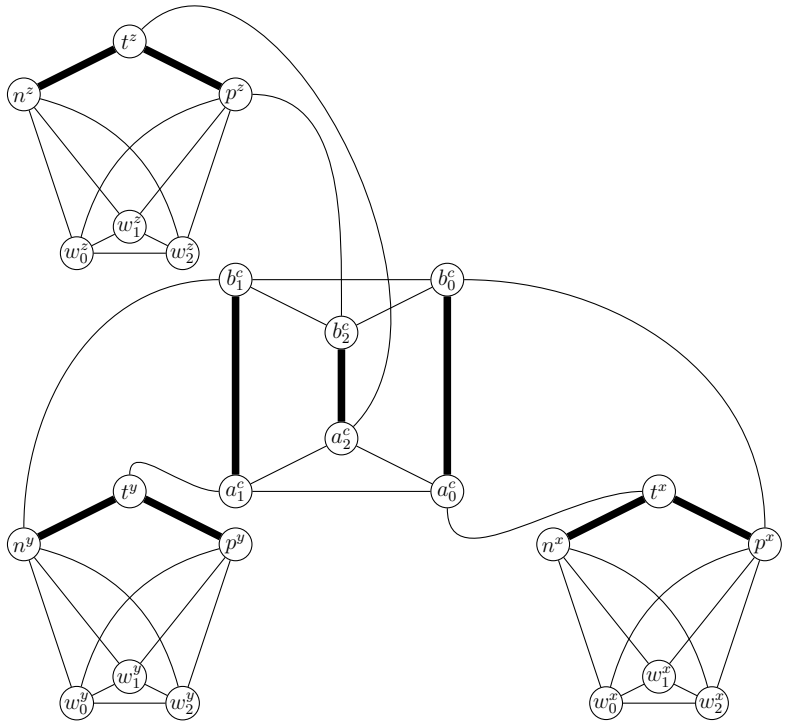


Figure 15.2: The connections for a clause $c = x \vee \neg y \vee z$. For the negated variable, $\neg y$, we connect the clause gadget to n^y and t^y, whereas for the variables in the non-negated form we have the clause connected to the t and p vertices. Observe that a budget of five is sufficient and necessary for eliminating all occurrences of C_4 in the depicted subgraph.

variable x appears in a non-negated form as the ith (for $i = 0, 1, 2$) variable in a clause c, we add the edges $t^x a_i^c$ and $p^x b_i^c$. If it appears in a negated form, we add the edges $t^x a_i^c$ and $n^x b_i^c$. The connections can be seen in Figure 15.2. Observe that we get exactly one extra induced C_4 in the connection, and that this can be eliminated by removing either one of the thick edges.

This concludes the construction. We have now obtained a graph G_φ constructed from an instance φ of 3SAT. We let $k_\varphi = |\mathcal{V}_\varphi| + 2|\mathcal{C}_\varphi|$ be the allowed (and necessary) budget, and the instance of C_4-FREE DELETION is then (G_φ, k_φ). The above exposition, with the tightness of the budget in mind, can be summarized in two observations.

Observation 15.1. *In any solution F of an instance (G_φ, k_φ),*

- *for each variable gadget G^x, exactly one of $n^x t^x$ and $t^x p^x$ is contained in F, and*
- *for each clause gadget G^c exactly two of the three edges $\{a_0^c b_0^c, a_1^c b_1^c, a_2^c b_2^c\}$ are in F.*

We now proceed to prove the following lemma, which will give the result.

Lemma 15.2. *A given* 3SAT *instance φ has a satisfying assignment if and only if (G_φ, k_φ) is a yes-instance of* C_4-FREE DELETION.

Proof. Let φ be satisfiable and G_φ and k_φ be as above. We show that (G_φ, k_φ) is a yes-instance for C_4-FREE DELETION. Let $\alpha\colon \mathcal{V}_\varphi \to \{\texttt{true}, \texttt{false}\}$ be a satisfying assignment for φ. We now define a solution F^α_φ. For every variable $x \in \mathcal{V}_\varphi$, if $\alpha(x) = \texttt{true}$, add to F^α_φ, from the variable gadget G_x, the edge corresponding to true, that is, the edge $t^x p^x$; otherwise add to F^α_φ the edge corresponding to false, that is, the edge $t^x n^x$. Every clause $c \in \mathcal{C}$ is satisfied by α; we pick an arbitrary variable x, say, the lexicographically first, whose literal satisfies c and add to F^α_φ two edges corresponding to the two other literals. If a clause is satisfied by more than one literal, we pick any of the corresponding variables.

For every clause, F^α_φ contains exactly two edges and for every variable exactly one edge. Thus the total number of edges in F^α_φ is $2|\mathcal{C}_\varphi| + |\mathcal{V}_\varphi| = k_\varphi$. We argue now that the graph $G^\alpha_\varphi = G_\varphi - F^\alpha_\varphi$ is C_4-free. Since variables appearing in clauses are pairwise different, it can be easily observed that every induced cycle of length four in G_φ is either

- entirely contained in some clause gadget, or
- entirely contained in some variable gadget, or
- is of form $t^x \gamma^x b^c_i a^c_i$, where x is the ith variable of clause c, and $\gamma \in \{n, p\}$ denotes whether the literal in c that corresponds to x is negated or non-negated.

By the construction of G^α_φ, we destroyed all induced 4-cycles of the first two types. Consider a 4-cycle $\langle t^x, p^x, b^c_i, a^c_i\rangle$ of the third type, where x appears positively in clause c. In the case when the literal of variable x was not chosen to satisfy c, we have deleted the edge $a^c_i b^c_i$ and so this 4-cycle is removed. Otherwise we have without loss of generality that $\alpha(x) = \texttt{true}$, and we have deleted the edge $t^x p^x$, thus also removing the considered 4-cycle. The case of a 4-cycle of the form $\langle t^x, n^x, b^c_i, a^c_i\rangle$ is symmetric.

Thus we get that all the induced 4-cycles that were contained in G_φ have been removed in G^α_φ. Since vertex pairs (a^c_i, b^c_i) and (γ^x, t^x) for $\gamma \in \{n, p\}$ do not have common neighbors, it follows that no new C_4 could be created when obtaining G^α_φ from G_φ by removing edges as described above. We infer that G^α_φ is indeed C_4-free.

We now prove the reverse direction. Let F be an edge set of G_φ of size at most k_φ such that $G_\varphi - F$ is C_4-free. By the definition of the budget k_φ and the observation that every variable gadget needs at least one edge to be in F and every clause gadget needs at least two edges to be in F (note here that the edge sets of clause and variable gadgets are pairwise disjoint), we have that F contains *exactly* one edge from each variable gadget, *exactly* two edges from each clause gadget, and no other edges.

We construct an assignment $\alpha\colon \mathcal{V}_\varphi \to \{\texttt{true}, \texttt{false}\}$ for the formula φ as follows. For a variable $x \in \mathcal{V}_\varphi$, we let $\alpha(x) = \texttt{true}$ if the true edge $t^x p^x$ of G^x is in F, and $\alpha(x) = \texttt{false}$ otherwise. By Observation 15.1, it follows that if $\alpha(x) = \texttt{false}$, then the false edge, $t^x n^x$ of G^x is in F. We claim that the assignment α satisfies φ.

Suppose for the sake of a contradiction that a clause $c \in \mathcal{C}$ is not satisfied. Since exactly two edges in the clause gadget G_c are in F, there is a variable x in c such that the corresponding variable-edge of G_c is not in F. If $\alpha(x) = \texttt{true}$, then because c is not satisfied, we have that $\neg x \in c$. By the definition of α we have that the false edge of G_x does not belong to F. Then in G_φ, the false edge of G_x and the variable-edge of G_c corresponding to x form part of an induced C_4 that is not destroyed by F, a contradiction. The case $\alpha(x) = \texttt{false}$ is symmetric. This concludes the proof of the lemma. □

Finally, the proof of Theorem 21 follows from Lemma 15.2: Combining the presented reduction with an algorithm for C_4-FREE DELETION working in $2^{o(k)}\,\mathrm{poly}(n)$ time would yield an algorithm which solves 3SAT in $2^{o(n+m)}$ time, which contradicts ETH by the results of Impagliazzo, Paturi and Zane [IPZ01]. Since $|V(G_\varphi)| = 6|\mathcal{V}_\varphi| + 6|\mathcal{C}_\varphi|$, and $|E(G_\varphi)| = 11|\mathcal{V}_\varphi| + 9|\mathcal{C}_\varphi| + 6|\mathcal{C}_\varphi|$, we also get lower bounds of the type $2^{o(n+m)}$ for C_4-FREE DELETION.

15.2 C_4-free completion

We mentioned in the part on subexponential time algorithms that it seems more completion problems have subexponential time algorithms more often than the deletion problems. So it could still be that the C_4-FREE COMPLETION could be solvable in subexponential time. Unfortunately, that is not the case as we now show. C_4-FREE COMPLETION is NP-complete and not solvable in subexponential time unless the exponential time hypothesis fails.

C_4-FREE COMPLETION parameterized by k

Input:	A graph G and an integer k
Question:	Does there exist a set F of at most k edges such that $G + F$ is C_4-free?

Theorem 22. *C_4-FREE COMPLETION, or $2K_2$-FREE DELETION is NP-complete and, under ETH, cannot be solved in time $2^{o(k)}\,\mathrm{poly}(n)$.*

The observant reader may observe that the theorem leaves out one thing several of the other theorems contained; We do not show lower bounds of the form $2^{o(n+m)}$. The reason, as will be clear during the construction, is that the clause gadget is not of constant size; the graph will have size $\Omega(|\mathcal{C}_\varphi|^2)$. It is not hard to imagine that using 4SAT instead, we could get similar lower bounds as before.

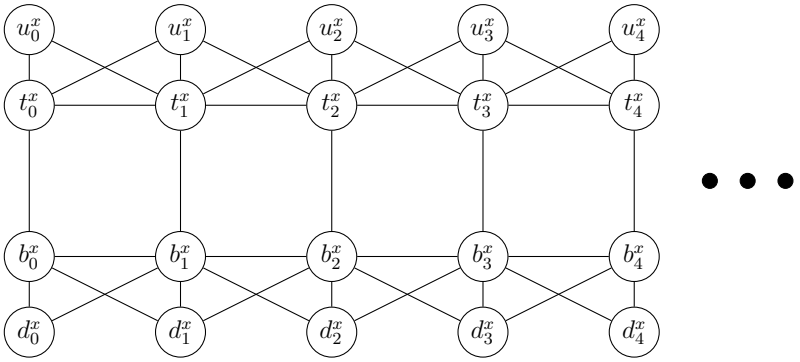
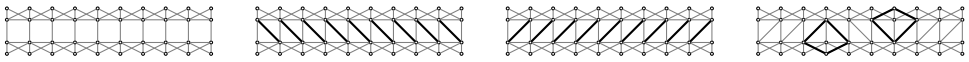


Figure 15.3: Variable gadget G_x.



(a) Before completion (b) True (c) False (d) Differing orientations

Figure 15.4: The variable gadget G_x, before completion, its two completions corresponding to assignments and a completion with differing orientations.

Again we reduce from 3SAT, and similarly as before we start with a formula where each clause contains exactly three literals corresponding to pairwise different variables. By duplicating clauses if necessary, we also assume that each variable appears in at least two clauses. Let φ be the 3SAT instance. We yet again need two types of gadgets, one gadget to emulate variables in the formula and one gadget to emulate clauses. We construct the graph G_φ as follows:

Variable gadget. For each variable $x \in \mathcal{V}_\varphi$ we construct a variable gadget graph G_x as depicted in Figure 15.3. Let p_x be the number of clauses x occurs in; by our assumption we have that $p_x \geq 2$. The graph G_x consists of a "tape" of $4p_x$ squares arranged in a cycle, with additional vertices attached to the sides of the tape. The intuition is that every fourth square in G_x is "reserved" for a clause in which x occurs. Formally, the vertex set of G_x is

$$V(G_x) = \bigcup_{0 \leq i < 4p_x} \{u_i^x, t_i^x, b_i^x, d_i^x\},$$

and the edge set is

$$E(G_x) = \bigcup_{0 \leq i < 4p_x} \{u_i^x t_i^x, u_i^x t_{i+1}^x, t_i^x u_{i+1}^x, t_i^x t_{i+1}^x, \\ t_i^x b_i^x, b_i^x b_{i+1}^x, b_i^x d_{i+1}^x, b_i^x d_i^x, d_i^x b_{i+1}^x\},$$

where the indices are taken modulo $4p_x$. The letters for the vertices are chosen to correspond with top and bottom (t^x and b^x) of tape, and up and down (u^x and d^x). The construction is visualized in Figures 15.3 and 15.4a.

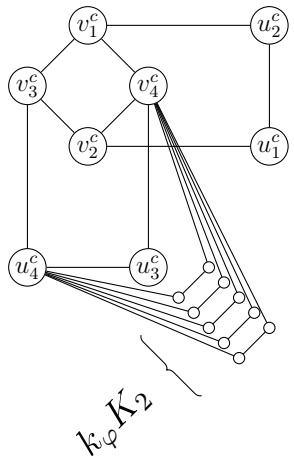


Figure 15.5: The clause gadget G_c. It contains one C_4, and if we add either edge $v_1^c v_2^c$ or edge $v_3^c v_4^c$, we get a new C_4 that must be destroyed by adding one more edge. The $k_\varphi K_2$ gadget makes sure we cannot add edge $v_4^c u_4^c$.

Claim 15.3. *The minimum number of edges required to add to G_x to make it C_4-free is $4p_x$. Moreover, there are exactly two ways of eliminating all 4-cycles with $4p_x$ edges, namely adding an edge on the diagonal for each square. Furthermore, if we add one edge to eliminate some cycle, all the rest must have the same orientation, i.e., all added edges are either of the form $t_i^x b_{i+1}^x$ or of the form $t_{i+1}^x b_i^x$. See Figure 15.4.*

Proof of claim. A gadget G_x contains $4p_x$ induced four-cycles, and no two of them can be eliminated by adding just one edge. Hence, to eliminate all four-cycles in G_x, we need at least $4p_x$ edges. On the other hand, it is easy to verify that after adding $4p_x$ diagonals to four-cycles of the same orientation the resulting graph does not contain any induced C_4, see Figure 15.4 for examples. Whenever we have two consecutive cycles with completion edges of different orientation, we create a new C_4 consisting of the two completion edges, and (depending on their orientation) either two edges incident to vertex u_i^x above their common vertex, or two edges incident to vertex d_i^x below. See Figure 15.4d. ⌟

Corollary 15.4. *The minimum number of edges required to eliminate all C_4s appearing inside all the variable gadgets is $12|\mathcal{C}_\varphi|$.*

Proof. Since each clause of $\mathcal{C}_\varphi$ contains exactly three occurrences of variables, it follows that $\sum_{x \in \mathcal{V}_\varphi} p_x = 3|\mathcal{C}_\varphi|$. The constructed variable gadgets are pairwise disjoint, so by Claim 15.3 we infer that the minimum number of edges required in all the variable gadgets is equal to $\sum_{x \in \mathcal{V}_\varphi} 4p_x = 3 \cdot 4|\mathcal{C}_\varphi| = 12|\mathcal{C}_\varphi|$. □

Clause gadget. We now proceed to create the clause gadgets. For each clause $c \in \mathcal{C}_\varphi$, we create the graph G_c as depicted in Figure 15.5. It consists of an induced 4-cycle $\langle v_1^c, v_4^c, v_2^c, v_3^c \rangle$ and induced paths $\langle v_2^c, u_1^c, u_2^c, v_1^c \rangle$ and $\langle v_3^c, u_4^c, u_3^c, v_4^c \rangle$.

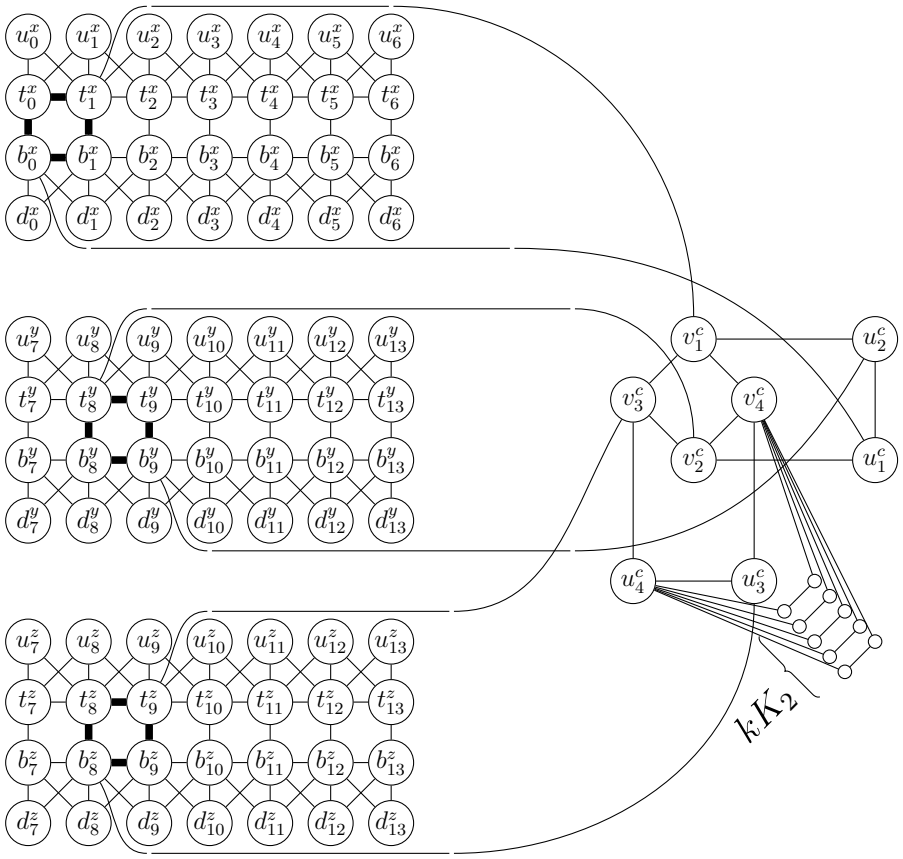
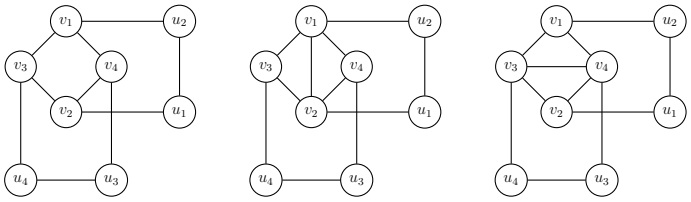


Figure 15.6: The connections for a clause $c = x \vee \neg y \vee z$. In this example, c is the first clause of appearance for x thus x is connected to G_c via the 0th square. For y and z, we assume that c is the third clause they appear, thus y and z use the 8th square.

We also attach a gadget consisting of k_φ internally disjoint induced paths of four vertices with endpoints in v_4^c and u_4^c, where k_φ is the budget to be specified later. This makes it impossible to add an edge between v_4^c and u_4^c in any C_4-free completion with at most k_φ edges.

By the i-th *square* we mean a four-cycle $\langle t_i^x, b_i^x, t_{i+1}^x, b_{i+1}^x \rangle$. If a clause c is the ℓ-th clause the variable x appears in, we will use the vertices of the $4(\ell-1)$-st square for connections to the gadget corresponding to c. For ease of notation let $j = 4(\ell-1)$. We also use pairs $\{v_1^c, u_1^c\}$, $\{v_2^c, u_2^c\}$, and $\{v_3^c, u_3^c\}$ of G_c for connecting to the corresponding variable gadgets. If a variable x appears in a non-negated form as the ith (for $i = 1, 2, 3$) literal of a clause c, then we add the edges $t_{j+1}^x v_i^c$ and $b_j^x u_i^c$. If it appears in a negated form, we add the edges $t_j^x v_i^c$ and $b_{j+1}^x u_i^c$. See Figure 15.6. This concludes the construction of G_φ. Finally, we set the budget for the instance to be $k_\varphi = 14|\mathcal{C}_\varphi|$.

Claim 15.5. *For each clause gadget G_c for a clause $c \in \mathcal{C}_\varphi$, we need to add at least* two edges *between vertices of G_c to eliminate all induced C_4s in G_c. Moreover, there are exactly three ways of adding exactly two edges to G_c so that the resulting*



(a) Before completion (b) Variable 1 or 2 in the clause must be satisfied (c) Variable 3 in the clause must be satisfied

Figure 15.7: The clause gadget

graph does not contain any induced C_4: by adding $\{v_1^c v_2^c, v_1^c u_1^c\}$, $\{v_1^c v_2^c, v_2^c u_2^c\}$, or $\{v_3^c v_4^c, v_3^c u_3^c\}$.

Proof of claim. There is a four-cycle $\langle v_1^c, v_4^c, v_2^c, v_3^c\rangle$ which needs to be eliminated, either by adding the edge $v_1^c v_2^c$ (Figure 15.7b) or $v_3^c v_4^c$ (Figure 15.7c). In any case we create a new C_4, either $\langle v_1^c, u_2^c, u_1^c, v_2^c\rangle$ in the former case, and $\langle v_4^c, u_3^c, u_4^c, v_3^c\rangle$ in the latter case. In the former case we can eliminate the created C_4 by adding $v_1^c u_1^c$ or $v_2^c u_2^c$, and in the latter case we can eliminate it by adding $v_3^c u_3^c$. Note that in the latter case we cannot add $v_4^c u_4^c$, since then we would create k_φ new induced four-cycles. A direct check shows that all the three aforementioned completion sets lead to a C_4-free graph. ⌟

Lemma 15.6. *A 3-CNF-SAT formula φ has a satisfying assignment if and only if (G_φ, k_φ) is a yes-instance of C_4-free Completion, where $k_\varphi = 14|\mathcal{C}_\varphi|$.*

Proof. In the forward direction, suppose φ is satisfiable with a satisfying assignment $\alpha\colon \mathcal{V}_\varphi \to \{\texttt{true}, \texttt{false}\}$. For every variable $x \in \mathcal{V}_\varphi$, if $\alpha(x) = \texttt{true}$, we add the edges $t_i^x b_{i+1}^x$ to the completion set F^α for $i \in \{0, \ldots, 4p_x - 1\}$ and if $\alpha(x) = \texttt{false}$, we add the edges $t_{i+1}^x b_i^x$ to F^α for $i \in \{0, \ldots, 4p_x - 1\}$.

For a clause c in $\mathcal{C}_\varphi$, if the first literal satisfies the clause, we add the edges $v_1^c v_2^c$ and $v_1^c u_1^c$ to F^α. If the second literal satisfies the clause, we add $v_1^c v_2^c$ and $v_2^c u_2^c$ to F^α and if it is the third literal, we add $v_3^c v_4^c$ and $v_3^c u_3^c$ to F^α. If more than one literal satisfies the clause, we pick any one. In total this makes $12|\mathcal{C}_\varphi|$ edges added to the variable gadgets and $2|\mathcal{C}_\varphi|$ edges added to the clause gadgets.

Suppose now for a contradiction that $G_\varphi + F^\alpha$ contains a cycle L of length four. From Claims 15.3 and 15.5 we know that L is not completely contained in a variable or clause gadget. Each vertex has at most one incident edge ending outside the gadget of the vertex and such edges are there only between variable and clause gadgets. Thus L consists of one edge from a variable gadget and one from a clause gadget and two edges between. We can observe that L then must contain either $v_1^c u_1^c$, $v_2^c u_2^c$, or $v_3^c u_3^c$ of the clause gadget, see Figure 15.6. Let us assume without loss of generality that L contains the edge $v_1^c u_1^c$. By the construction of the set F^α this implies that the literal of the first variable x of c satisfies c. If x is non-negated in c, then we have that $\alpha(x) = \texttt{true}$ and that $v_1^c t_{j+1}^x$ and $u_1^c b_j^x$ are edges of L. To complete the cycle $t_{j+1}^x b_j^x$ must be an edge of L; however, by the

definition of F^α we have added the edge $t_j^x b_{j+1}^x$ to F^α instead of $t_{j+1}^x b_j^x$, and we obtain a contradiction. The case where x is negated is symmetric.

In the reverse direction, suppose (G_φ, k_φ) is a yes-instance for $k_\varphi = 14|\mathcal{C}_\varphi|$ and let F be such that $G_\varphi + F$ is C_4-free with $|F| \leq k_\varphi$. By Corollary 15.4 and Claim 15.5 we know that we need to use at least $12|\mathcal{C}_\varphi|$ edges to fix the variable gadgets and we need to use at least $2|\mathcal{C}_\varphi|$ edges for the clause gadgets. Since $|F| \leq k_\varphi$, we infer that $|F| = k_\varphi$, that we use exactly $4p_x$ edges to fix each variable gadget G_x (and that the orientation of the added edges must be the same within each gadget), that we use exactly two edges for each clause gadget G_c, and that F contains no edges other than those mentioned above.

We now define an assignment α for $\mathcal{V}_\varphi$ and prove that it is indeed a satisfying assignment. If F contains the edge $t_0^x b_1^x$, we let $\alpha(x) = \texttt{true}$, and if F contains the edge $t_1^x b_0^x$ we let $\alpha(x) = \texttt{false}$. Let $c \in \mathcal{C}_\varphi$ be a clause and suppose that c is not satisfied by the assignment α. We know by Claim 15.5 that the gadget for c contains $\{v_1^c v_2^c, v_1^c u_1^c\}$, $\{v_1^c v_2^c, v_2^c u_2^c\}$, or $\{v_3^c v_4^c, v_3^c u_3^c\}$.

Without loss of generality assume that G_c contains $\{v_1^c v_2^c, v_1^c u_1^c\}$ and that x is the first variable in c, and that it appears non-negated. Since x does not satisfy c, we infer that $\alpha(x) = \texttt{false}$. This means that $t_1^x b_0^x \in F$, and since the orientation of the added edges in the gadget G_x is the same, then also $t_{i+1}^x b_i^x \in F$. As a result, both edges $t_{i+1}^x b_i^x$ and $v_1^c u_1^c$ are present in $G_\varphi + F$. But then we have an induced four-cycle $\langle v_1^c, u_1^c, b_i^x, t_{i+1}^x, v_1^c \rangle$, contradicting the assumption that $G_\varphi + F$ was C_4-free. The cases for y, z and negative literals are symmetric. This concludes the proof. □

Similarly as before, the proof of Theorem 22 can be completed as follows: combining the presented reduction with an algorithm which solves C_4-free Completion in $2^{o(k)} \cdot \text{poly}(n)$ time would give an algorithm which solves 3Sat in $2^{o(n+m)}$ time, which contradicts ETH by the results of Impagliazzo, Paturi, and Zane [IPZ01].

Chapter 16

Biclusters and starforests

As promised in the part on subexponential time algorithms, we show that if we do not bound the number of connected components in the target graph—we refer to this bound as p—there is little chance of obtaining subexponential time algorithms for the problems Starforest Editing and the more general Bicluster Editing. When we have a bound on the number of connected components in the target graph, we refer to these problems as p-Starforest Editing and p-Bicluster Editing. We finally show that parameterizing by both p and k is necessary; Parameterized by p alone, it turns out that p-Starforest Editing is W[1]-hard.

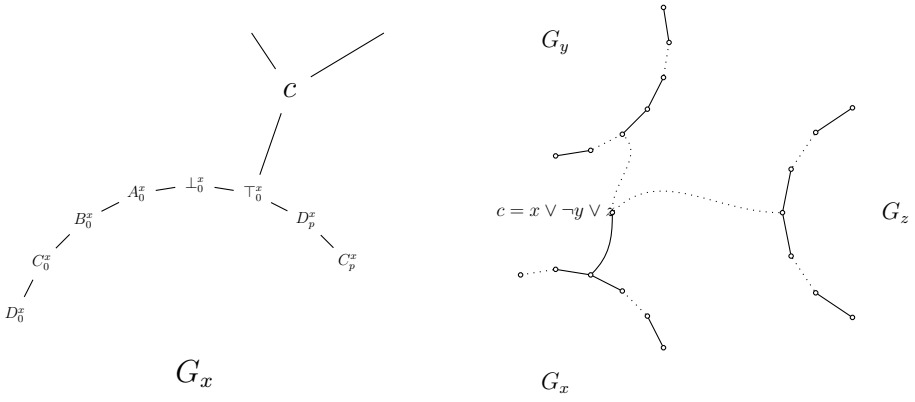
16.1 Hardness for bicluster editing

We first show that Starforest Editing is NP-hard and that we cannot expect a subexponential algorithm unless the exponential time hypothesis fails. We again (for the last time) describe a linear reduction from 3Sat. This time to Starforest Editing. Furthermore, the instance we reduce to has maximal degree three, thus not only showing that Starforest Editing is NP-complete on graphs of bounded degree, but also not solvable in subexponential time on subcubic graphs.

Theorem 23. *The problem* Starforest Editing *is* NP*-complete and, assuming ETH, does not admit a subexponential parameterized algorithm when parameterized by the solution size k, i.e., it cannot be solved in time $2^{o(k)}\operatorname{poly}(n)$, nor in exact exponential time $2^{o(n+m)}$, even when restricted to subcubic graphs.*

We resort to similar reductions as used in Chapter 14, but the gadgets here are even simpler. They are also subcubic, so we achieve lower bounds for subcubic graphs. In Chapter 14 we needed vertices of degree four, but here we manage to push the degree to three. See Figures 16.1a and 16.1b for figures of the gadgets.

Variable gadget. We construct for $x \in \mathcal{V}_\varphi$ a graph $G_x \cong C_{6p_x}$ where p_x is the number of clauses in φ which x appears in. The vertices of G_x are labeled, consecutively, $\top_i^x, \bot_i^x, A_i^x, B_i^x, C_i^x, D_i^x$ for $i \in [0, p_x - 1]$.



(a) Parts of a variable gadget x and its connection when occurring positively in c.

(b) Deletion when x is chosen to satisfy the clause. If we chose not to delete the edge connecting the clause vertex with G_y we would have gotten an induced P_4.

Figure 16.1: Reduction from 3Sat to Starforest Editing on subcubic graphs

There are exactly three ways of deleting G_x into a starforest using at most $k_x = 6p_x$ edges. Clearly a collection of P_3s is a starforest and is our target graph. We will define the $\top$-deletion for G_x as the deletion set $F_\top^x = \{C_i^x D_i^x, \perp_i^x A_i^x \mid i \leq p_x - 1\}$ and the $\perp$-deletion for G_x as the deletion set $F_\perp^x = \{A_i^x B_i^x, D_i^x \top_{i+1}^x \mid i \leq p_x - 1\}$, taking the $i+1$ in the index of $\top_{i+1}^x$ modulo p_x. In other words, in the gadget G_x, we are *keeping* the edges in

- $\langle D_{i-1}^x, \top_i^x, \perp_i^x \rangle$ and $\langle A_i^x, B_i^x, C_i^x \rangle$, when x is set to true, and
- $\langle \top_i^x, \perp_i^x, A_i^x \rangle$ and $\langle B_i^x, C_i^x, D_i^x \rangle$, when x is set to false.

Observe that when x is set to true, we will have paths on three vertices, where $\top_i^x$ is the middle vertex, and if x is set to false, we will have paths on three vertices with $\perp_i^x$ being the middle vertex. Later, we will see that if x satisfies a clause c, the ith clause x appears in, then either $\top_i^x$ or $\perp_i^x$ will be the middle vertex of a claw, depending on whether x appears positively or negatively in c.

Observation 16.1. *In an optimal edge edit of a cycle of length divisible by* 6*, no edge is added and exactly every third consecutive edge is deleted.*

Clause gadget. A clause gadget simply consists of one vertex, i.e., for a clause $c \in \mathcal{C}_\varphi$, we construct the vertex v_c. This vertex will be connected to G_x, G_y and G_z, for x, y, z being its variables, in appropriate places, depending on whether or not the variable occurs negated in c. In fact, it will be connected to $\top_i^x$ if c is the ith clause x appears in, and x appears positively in c, and it is connected to $\perp_i^x$ if c is the ith clause x appears in, and x appears negatively in c.

Let $k_\varphi = 2|\mathcal{C}| + 2\sum_x p_x = 2|\mathcal{C}| + 3 \cdot 2|\mathcal{C}| = 8|\mathcal{C}|$ be the budget for a formula φ. We now observe that the budget is tight.

Lemma 16.2. *The graph G_φ has no starforest editing set of size less than k_φ, and if the editing set has size k_φ it contains only deletions.*

Proof. It is straightforward to verify that for each induced variable gadget G_x we need at least $2p_x$ edges. Since every clause contains three variables, we have $|\mathcal{C}|/3$ such gadgets, and their necessary budget sum up to exactly $\sum_x 2p_x \cdot 3 = 6|\mathcal{C}|$.

Since no two consecutive edges in G_x will be deleted, by the previous observation, we have that for each clause, after deleting edges in the variable gadgets, we will have an induced subdivided claw with the clause vertex as its center, and this graph needs at least two edits to become a star forest. This can be verified by observing that we have three induced P_5s, and at most two of them can be removed by one edge edit.

From the above analysis, we can conclude that G_φ needs at least $6|\mathcal{C}| \cdot 2|\mathcal{C}| = 8|\mathcal{C}| = k_\varphi$ edits to become a starforest graph. □

We now continue to the main lemma, from which Theorem 23 follows.

Lemma 16.3. *A* 3Sat *instance φ is satisfiable if and only if (G_φ, k_φ) is a yes-instance of* Starforest Editing.

Proof. Suppose φ was satisfiable and let $\alpha\colon \mathcal{V}_\varphi \to \{\texttt{false}, \texttt{true}\}$ be a satisfying assignment. We show that $G - F^\alpha$ for F^α defined below is a starforest graph and that $|F^\alpha| \leq k_\varphi$ (since the budget is tight, we have equality). For $x \in \mathcal{V}_\varphi$ we define F^α_x to be the following set of edges:

- $F^\alpha_x = \{C^x_i D^x_i, \bot^x_i A^x_i \mid i \leq p_x - 1\}$, if $\alpha(x) = \texttt{true}$.
- $F^\alpha_x = \{A^x_i B^x_i, D^x_i \top^x_{i+1} \mid i \leq p_x - 1\}$, if $\alpha(x) = \texttt{false}$.

Finally, for a clause $c \in \mathcal{C}_\varphi$, let x_c be a variable satisfying c. Define F^α_c to be the two edges not incident to x_c.

We now show that $F^\alpha = \bigcup_{x\in\mathcal{V}} F^\alpha_x \cup \bigcup_{c\in\mathcal{C}} F^\alpha_c$ is our solution. It should at this point be clear that $|F^\alpha| \leq k_\varphi$. Since $G_x - F^\alpha_x$ is a collection of P_3s, we only need to verify that no clause gadget c is in an obstruction. Let c be an arbitrary clause gadget and let x_c be the variable that is still incident to c. Clearly, since c is of degree 1, it has to be in an obstruction with x_c. However, since x_c satisfies c, and (for the moment) assuming that x_c appears positively in c, $\alpha(x) = \texttt{true}$ and from G_x, we deleted $C^x_i D^x_i$ and $\bot^x_i A^x_i$ for every i. Since c connects to some vertex $\top^x_i$, the connected component containing c is a claw centered in $\top^x_i$ with leaves c, D^x_i and $\bot^i_x$. Hence c cannot be in an obstruction. The case when x_c appears negatively is symmetric. This concludes the forward direction of the proof.

For the reverse direction, suppose (G_φ, k_φ) is a yes-instance of Starforest Editing and let F be a solution. Since the budget is tight, by the above lemma and observation, we know that F contains only deletions. There are unique ways of deleting all the G_xs for the variable gadgets, so construct an assignment for the variables of φ, $\alpha_F\colon \mathcal{V}_\varphi \to \{\texttt{false}, \texttt{true}\}$ by letting $\alpha_F(x) = \texttt{true}$ if for some i, the edge $\bot^x_i A^x_i$ is deleted, and let $\alpha_F(x) = \texttt{false}$ otherwise. We claim that α_F

is a satisfying assignment. Suppose that a clause c is not satisfied by any of its variables, and consider x_c, the variable c is still adjacent to. We know it must be adjacent to at least one vertex since the budget is tight (not all three edges were deleted). Suppose x_c appeared positively in c (thus the vertex for c is adjacent to some $\top_i^x$). Since $G - F$ is a starforest (recall that F can only contain deletions in the given budget), we know that in the subgraph G_x to which x_c belongs, we must have deleted the edge $\bot_i^x A_i^x$, for otherwise, since every third edge is deleted, the edges $D_{i-1}^x \top_i^x$ and $C_{i-1}^x D_i^x$ would remain and form an induced P_4, contradicting the assumption that $G - F$ was a starforest graph. But since $\bot_i^x A_i^x$ was deleted, by the construction of α_F, we set x to true, so x indeed satisfies c contradicting the initial assumption. The case where x_c appears negatively in c is symmetric. □

Observing that the maximum degree of G_φ is three—the clause vertices have exactly degree three, and the variable gadgets are cycles with some vertices connected to at most one clause vertex—this concludes the proof of Theorem 23. From the discussions above, the following result is an immediate consequence:

Corollary 16.4. *The problem* Starforest Deletion *is* NP*-complete and not solvable in subexponential time under ETH, even on subcubic graphs.*

Before going into parameterized lower bounds of Starforest Editing, we show that the exact same reduction above simultaneously prove similar results for the bicluster case. We note that the NP-hardness was shown by Amit [Ami04], but their reduction suffers a quadratic blowup and is therefore not suitable for showing subexponential lower bounds.

Corollary 16.5. *The problems* Bicluster Editing *and* Bicluster Deletion *are* NP*-complete and not solvable in subexponential time under ETH, even on subcubic graphs.*

Proof. We show that every optimum solution of (G_φ, k_φ) for the above constructed G_φ and k_φ will yield a starforest and hence the corollary follows from the above result. We first show that Observation 16.1 also holds for the Bicluster Editing case, that is, for budget k_x, there is a unique (up to rotation) solution which consists of deleting every third edge. First, we observe that deleting every third edge indeed is a solution as starforests are a subclass of biclusters. Second, we can pack $3p_x$ paths of length four such that each pair of P_4s share at most one edge, and such that any edit can eliminate at most two obstructions. Hence we need at least $2 \cdot 3p_x = k_x$ edges to eliminate all the P_4s. Since the budget is tight for G_x, we now show that we still need at least two edges to eliminate G_c

Consider a clause-gadget. Since we have the same situation as above, i.e. it contains three induced P_5s, we observe that at least two edits inside the gadget are necessary. Suppose that one of the edits is an edge addition (needed to make a biclique that is not a star), then we must use that edge to construct a C_4. But this edit leaves one induced P_5 which cannot be resolved by the remaining edit.

By combining the arguments for G_x and G_c, we conclude that (G_φ, k_φ) is a yes-instance if and only if φ is satisfiable and furthermore that the solution will only delete edges, thus yielding a starforest. □

16.2 W[1]-hardness parameterized by stars

In this section we show that the parameterization by k is necessary, even for the case of p-Starforest Editing. That is, we show that when we parameterize by p alone, the problem becomes W[1]-hard, and we can thus not expect any algorithms of running time $f(p) \cdot \text{poly}(n)$ for any function f solving p-Starforest Editing.

p-Starforest Editing parameterized by p

Input:	A graph $G = (V, E)$ and a non-negative integer k.
Question:	Is there a set of at most k edges F such that $G \triangle F$ is a disjoint union of stars?

We reduce from the problem Multicolored Regular Independent Set. An instance of this problem consists of a regular graph colored into p color classes, each color class inducing a complete graph, and we are asked to find an independent set of size p.

Proposition 16.6 ([CFK+15])**.** *The problem* Multicolored Regular Independent Set *is* W[1]*-complete.*

Since each color class is complete, any independent set will be of size at most p and any independent set of size p is maximum. The reduction is direct, in fact we have that given a budget $k = (n - p)(d - 1)$, where d is the regularity degree, the following direct translation between the two problems holds:

Lemma 16.7. *Let G be a d-regular graph on n vertices, $p \leq n$, and fix the budget $k = (n - p)(d - 1)$. Then (G, p) is a yes-instance of* Multicolored Regular Independent Set *if and only if (G, k) is a yes-instance of p-*Starforest Editing*.*

Proof. In the forward direction, suppose S is an independent set of size p in G. Then, since S is maximal, every vertex in $G - S$ is adjacent to S. For every vertex $v \notin S$, delete $d - 1$ edges, but keep one connected to a vertex in S. Since there are $n - p$ vertices outside S, and since S is an independent set, this is exactly all the edges we need keep and we obtain a starforest editing with exactly budget k.

For the reverse direction, let us assume that G does not contain an independent set of size p. Hence, any set of p centers contains at least one edge; the total budget needed to edit to a starforest is then at least $(n - p)(d - 1) + 1 > k$ and hence the answer for (G, k) is no, as well. □

Combining Proposition 16.6 with Lemma 16.7 yields the following result:

Theorem 24. *p-*Starforest Editing *is* W[1]*-hard when parameterized by p.*

Part V

Concluding remarks

Chapter 17

Conclusions and future directions

In this chapter we conclude this thesis by stating some open problems and some research questions for the future. A summary of all the results in this thesis was listed in Section 1.6.2. The results following Lewis and Yannakakis' work on the complexity of vertex deletion problem [LY80] as well as Yannakakis' work on the Minimum Fill-In problem [Yan81a], combined with Fomin and Villanger's proof that Minimum Fill-In is solvable in subexponential parameterized time [FV13], have steadily been steered towards some new lines of research stated below. Some of the main research questions following this thesis are:

- ⋆ How far can we push subexponentiality beyond chordal graphs?
- ⋆ Which graph modification problems admit subexponential parameterized time algorithms?
- ⋆ Which graph modification problems admit polynomial kernels?

Future directions

Complexity dichotomies for $\mathcal{H}$-free Completion

Can we come up with general classifications for $\mathcal{H}$, possibly finite, for which

- $\mathcal{H}$-free Completion is NP-complete or in P,
- $\mathcal{H}$-free Completion admits a polynomial kernel,
- $\mathcal{H}$-free Completion is solvable in subexponential parameterized time,

and similar questions for $\mathcal{H}$-free Editing. Some results have recently started appearing. In a series of preprints, Aravind, Sandeep, and Sivadasan announced that H-free Deletion, Completion, and Editing is NP-complete if H is a graph with at least two edges, at least two non-edges, and at least three vertices, respectively [ASS15a, ASS15b]. This is adding one step towards a better understanding of the complexity landscape of these problems. For the second

question, many results were achieved by Cai [Cai12] in his Master thesis, which can be considered a continuation of the work by Guillemot et al. [GHPP13] and Kratsch and Wahlström [KW13]. For the third question, however, absolutely no dichotomies or general theorems are known except for the large classes of graphs we know do not have subexponential parameterized time algorithms according to ETH. In the aforementioned preprint by Aravind et al. [ASS15a], they do however show that for the problems mentioned there, a $2^{o(k)}\operatorname{poly}(n)$ lower bound under ETH should follow. Unfortunately, though, we are nowhere near a broad understanding of this complexity landscape.

Subexponentiality depends on kernelizability

It seems, with the only *possible* exception of Interval Completion, that every graph modification problem solvable in subexponential parameterized time admits a polynomial kernel. Indeed, in many cases it is a prerequisite for obtaining the algorithm. Most subexponential parameterized time algorithms of this kind consists of enumerating sublinear sized sets of vertices, however, to avoid getting XP-type running times, the domain over which we enumerate must be of size bounded polynomially in k. In the algorithm for Interval Completion, the authors get away without a polynomial kernel by only looking at small enough portions of the input graph at the same time [BFPP16].

Since Cluster Deletion admits a polynomial kernel [CM12], but does by the lower bounds result of Komusiewicz and Uhlmann [KU12] not have a subexponential parameterized time algorithm, the reverse direction does not hold. It is also easy to come up with problems for which there are subexponential parameterized time algorithms, but no polynomial kernels, unless the polynomial hierarchy collapses, but these problems are typically not $\mathcal{H}$-free Completions. Consider the following problem, which we slightly abusively call an Or-$\mathcal{H}$-free Completion problem:

OR-Trivially Perfect Completion parameterized by k

Input:	A graph G and an integer k
Question:	Does G have a connected component C such that (C, k) is a yes-instance of Trivially Perfect Completion?

This problem trivially cross-composes and is clearly NP-complete, so it cannot have a polynomial kernel unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. Furthermore, running Trivially Perfect Completion component-wise on the input graph G solves the problem in subexponential parameterized time.

The square root phenomenon: Chordal graphs

All our results on subexponential parameterized time algorithms have been for graph classes closely related to chordal graphs, that is, every graph class for which modifying towards have been subclasses of chordal graphs. There is a minor exception in the case of pseudosplit graphs. The pseudosplit graphs (see Section 2.2, Proposition 2.19), are not chordal, however, they can be represented as split graphs, which are chordal, plus n extra bits of information.

There is another exception in the problem p-Bicluster Editing. Recall that for a fixed p, we can solve this problem in $2^{O(\sqrt{k}\log k)} + \text{poly}(n)$ time (Theorem 14). However, this graph class is neither chordal nor co-chordal since C_4 is a bicluster and since $2K_2 = \overline{C_4}$ is a bicluster. But, here the situation may still be salvaged by arguing that p-Bicluster Editing is not definable as an $\mathcal{H}$-free Editing problem. The graph class p-bicluster—contrary to the class of p-cluster graphs—is not hereditary; removing the center vertex of a 1-starforest may result in a graph with arbitrarily many components.

So the interesting question is, for $\mathcal{H}$-free Editing and $\mathcal{H}$-free Completion, are there $\mathcal{H}$-free graph classes unrelated to chordal graphs for which any of the two problems above are NP-complete and solvable in subexponential parameterized time? This question is also related to the third complexity dichotomy for $\mathcal{H}$; for which $\mathcal{H}$ is $\mathcal{H}$-free Completion and $\mathcal{H}$-free Editing solvable in subexponential parameterized time.

Subexponential for targets with few components

As we have discussed in several occasions throughout this thesis (see e.g. Section 1.5), Fomin et al. [FKP+14], discovered that upon bounding the number of connected components in the target graph by p, they could obtain subexponential time algorithms for the problem p-Cluster Editing. Such results are ruled out for Cluster Editing under ETH. Misra et al. [MPS13] showed that this parameter does not always help in the aim of obtaining subexponential time algorithm; it does not for p-Club d-Cluster Editing. We saw also in Chapters 12.1 and 16.1 that it *does* help for Bicluster Editing. Allowing arbitrarily many components in the target graph gives $2^{o(k)}\,\text{poly}(n)$ lower bounds under ETH, whereas bounding this number gives $2^{O(p\sqrt{k}\log(pk))} + \text{poly}(n)$ results.

Now, it is clear that bounding the number of components in the target for trivially perfect graphs does not help; If a graph class is closed under adding universal vertices, by adding a universal vertex to an input instance, any optimal solution will always be connected. An interesting question is to figure out exactly when bounding the number of connected components in the solution helps, and when not.

Some concrete important open problems

Classical complexity

Open problem 1. *What are sufficient and necessary conditions for* $\mathcal{H}$-free Deletion *to be* NP*-hard for finite* $\mathcal{H}$*?*

Open problem 2. *What are sufficient and necessary conditions for* $\mathcal{H}$-free Editing *to be* NP*-hard for finite* $\mathcal{H}$*?*

Here, we should always keep in mind that (Pseudo)Split Editing is polynomial time solvable [HS81].

Open problem 3. *Is* Pseudosplit Editing *solvable in linear time?*

One place to start is to see whether one can read directly from the degree sequence the C, S, I-partitioning of the closest solution.

Polynomial kernels

Claw-free and line graphs. From the Master thesis of Cai [Cai12], some of the interesting remaining open questions are $\mathcal{H}$-free Deletion for stars. Denote by S_n the star on n vertices, i.e., $K_{1,n-1}$. It is well known that $S_3 = K_{1,2} = P_3$ does admit a $2k$ vertex kernel, being the Cluster Deletion problem. On the other side, Cai [Cai12] showed that S_{11}-free Deletion does not admit a polynomial kernel unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Hence there is a large gap for H being S_4, the claw, up to S_{10}. Closing this gap is an interesting theoretical question. Recall that P_ℓ and C_ℓ have the corresponding edge modification problems admitting polynomial kernels if and only if the forbidden structure has at most three edges. Does the same hold for the stars?

Open problem 4. *Does* claw-free Deletion *admit a polynomial kernel?*

In a recent article, Cygan et al. [CPP+15] show, using an overall strategy similar to what we did for Trivially Perfect Editing, that deleting to {claw,diamond}-free graphs admits a polynomial kernel. This might seem like an arbitrary graph class but is not; These are the line graphs of triangle-free graphs [MT03, KS12]. This is also, as they argue, a first important step towards deletion to claw-free graphs; They highlight the next obstacle for obtaining a polynomial kernel (X is here a vertex modulator):

> Thus, we believe that for the sake of showing a polynomial kernel for claw-free Deletion, one needs to understand the three special cases when $G - X$ is (a) a line graph, (b) a proper interval graph, and (c) a co-bipartite graph. [CPP+15]

Aravind and Sandeep [SS15] improve the size of the polynomial kernel of DIAMOND-FREE DELETION by Cai [Cai12], and ask

Open problem 5. *Does* PAW-FREE DELETION *admit a polynomial kernel?*

See Table 2.1 for definitions of the paw and the diamond. The paw is a claw plus an edge, and the diamond is a K_4 minus an edge. A graph G is a paw-free graph if and only if each component of G is triangle-free or complete multipartite [Ola88].

An interesting first attempt to tighten the bound between S_3 and S_{11} might be S_7-FREE DELETION. The reason why this might be interesting comes from considering the graph H_{KW}, which is obtained from S_7 by adding two non-incident edges. The problem H_{KW}-FREE DELETION does not admit a polynomial kernel unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ [KW13].

Open problem 6. *Can we show that* S_7-FREE DELETION *does not admit a polynomial kernel unless* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$*?*

A graph class slightly related to claw-free graphs are the line graphs. The class of *line graph* is characterized by nine forbidden induced subgraphs, the claw being one of them.

Open problem 7. *Does* LINE DELETION *admit a polynomial kernel?*

Interval graphs. The question of the parameterized complexity of INTERVAL COMPLETION was first asked in a FOCS paper in 1994 and then in a journal version by the same authors, Kaplan, Shamir, and Tarjan in 1999 [KST99]. This was answered in the positive more than a decade later by Villanger, Heggernes, Paul, and Telle [VHPT09]. In 2014, Bliznets, Fomin, Pilipczuk, and Pilipczuk showed that INTERVAL COMPLETION is solvable in subexponential parameterized time [BFPP16]. This is the first subexponential parameterized time algorithm for such a problem given without relying on a polynomial kernelization. To this date the kernelization complexity of this problem is wide open.

Open problem 8. *Does* INTERVAL COMPLETION *admit a polynomial kernel.*

There is currently a large gap in our knowledge of CHORDAL EDITING. The problem was just recently proved to be FPT by Cao and Marx [CM14], but they give a k^k-type algorithm. In Section 13.2.2, we saw a lower bound of the type $2^{o(\sqrt{k})} \cdot \text{poly}(n)$ using ETH. There is a big gap between

$$2^{O(k \log k)} \text{poly}(n) \text{ and } 2^{o(\sqrt{k})} \text{poly}(n).$$

What is the correct bound? Is it $2^{\Theta(k)} \text{poly}(n)$? In addition to the question in light of the optimality programme question, there is a question of whether the problem admits a polynomial kernel.

Open problem 9. *Does* Chordal Editing *admit a polynomial kernel?*

Finally, the following problem is still a valid and interesting problem, despite that we know that the problems have co-RP kernels [KW14] (see Section 1.4). Providing randomized kernels demanded completely new tools for these problems, namely applying matroid theory for preprocessing. This has already sparked new research directions, and there is a hope that the same thing may happen for the deterministic version.

Open problem 10. *Do* Odd Cycle Transversal *and* Edge Bipartization *admit deterministic polynomial kernels?*

Subexponential time

Of all the $\mathcal{H}$-free Completion problems we know to be solvable in subexponential parameterized time, only one is known to be solvable in time $2^{O(\sqrt{k})}\operatorname{poly}(n)$: Split Completion [CFK+15]. In addition, only one is not known to be solvable in time $2^{O(\sqrt{k}\log k)}\operatorname{poly}(n)$, namely Proper Interval Completion. A natural next step in the optimality programme is to improve the $2^{O(k^{2/3}\log k)}+\operatorname{poly}(n)$ algorithm by Bliznets et al. [BFPP14] to an algorithm running in time $2^{O(\sqrt{k}\log k)}\operatorname{poly}(n)$. In light of the optimality programme and the few lower bounds we have for these graph problems, a most interesting question is "what are the correct running times?"

As stated above, Fomin and Villanger [FV13] noticed that under ETH, we cannot solve Chordal Completion in time $2^{O(k^{1/6})}\operatorname{poly}(n)$. Bliznets et al. recently improved this to a lower bound on the form

$$2^{O(k^{1/4}/\log^c k)}\cdot\operatorname{poly}(n),$$

for some integer c, simultaneously providing lower bounds for the problems Interval Completion, Proper Interval Completion, Threshold Completion, Chain Completion, and Trivially Perfect Completion [BCK+16]. Under the assumption that there is no subexponential-time approximation scheme for Min Bisection on d-regular graphs, they give stronger bounds of the form $2^{o(\sqrt{k})}\operatorname{poly}(n)$ for all of those problems as well.

Open problem 11. *Is* Chordal Completion *solvable in* $2^{O(\sqrt{k})}\cdot\operatorname{poly}(n)$ *time?*

Open problem 12. *Does there exist a set* $\mathcal{H}$ *such that* $\mathcal{H}$-free Completion *is solvable in* $2^{o(k)}\cdot\operatorname{poly}(n)$ *but does not admit a polynomial kernel?*

Recall from above that there are plenty of problems that are solvable in subexponential parameterized time but does not admit a polynomial kernel—trivially cross composable problems like Planar k-path [DFHT05, BDFH09], or any problem of the form "does the input graph have a connected component C which is lacking k edges from being, say, a chordal graph?"

Going beyond chordal graphs. The *threshold signed* graphs are exactly the graphs of dilworth number 2, whereas threshold graphs are graphs of dilworth number 1. It is the first next level of natural generalizations of threshold graphs. The definition is the following:

Definition 17.1 (Threshold signed graph)**.** A graph is a *threshold signed* graph if there are two positive reals $s \in \mathbb{R}$ and $t \in \mathbb{R}$ and a weight function $w : V \to \mathbb{R}$, such that for every vertex v, $w(v) \leq \min\{s, t\}$ and for every two vertices u, v, $uv \in E(G)$ if and only if $|w(u) + w(v)| \geq s$ or $|w(u) - w(v)| \geq t$.

Open problem 13. *Is* Threshold Signed Completion FPT*? Is it solvable in subexponential parameterized time?*

The permutation graphs are a superclass of the threshold signed graphs, and are best definable using a geometric intersection model, like the interval graphs (Section 2.2).

Definition 17.2 (Permutation graph)**.** A graph is a permutation graph if and only if it has an intersection model consisting of straight lines between two parallel lines [BLS99].

Open problem 14. *Is* Permutation Completion FPT*? Is it solvable in subexponential parameterized time?*

A graph is a circle-polygon graph if it is the intersection graph of convex polygons inscribed in a circle.

Open problem 15. *Is* circle-polygon Completion FPT*? Is it solvable in subexponential parameterized time?*

Index

Bibliography

[AK10] F. N. Abu-Khzam. A kernelization algorithm for d-Hitting Set. *J. Comput. Syst. Sci.*, 76(7):524–531, 2010.

[AK15] F. N. Abu-Khzam. Private communication, 2015.

[ALS09] N. Alon, D. Lokshtanov, and S. Saurabh. Fast FAST. In *ICALP*, volume 5555 of *Lecture Notes in Computer Science*, pages 49–58. Springer, 2009.

[Ami04] N. Amit. *The bicluster graph editing problem.* PhD thesis, Tel Aviv University, 2004.

[AS09] N. Alon and U. Stav. Hardness of edge-modification problems. *Theoretical Computer Science*, 410(47-49):4920–4927, 2009.

[ASS14] N. R. Aravind, R. B. Sandeep, and N. Sivadasan. On polynomial kernelization of $\mathcal{H}$-free edge deletion. In *IPEC*, 2014.

[ASS15a] N. R. Aravind, R. B. Sandeep, and N. Sivadasan. Parameterized lower bound and NP-completeness of some H-free edge deletion problems. *CoRR*, abs/1507.06341, 2015.

[ASS15b] N. R. Aravind, R. B. Sandeep, and N. Sivadasan. Parameterized lower bounds and dichotomy results for the NP-completeness of H-free edge modification problems. *CoRR*, abs/1509.08807, 2015.

[BBC04] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1):89–113, 2004.

[BBD06] P. Burzyn, F. Bonomo, and G. Durán. NP-completeness results for edge modification problems. *Discrete Applied Mathematics*, 154(13):1824–1844, 2006.

[BCC+06] H. L. Bodlaender, L. Cai, J. Chen, M. R. Fellows, J. A. Telle, and D. Marx. Open problems in parameterized and exact computation. In *IWPEC*, 2006.

[BCK+16] I. Bliznets, M. Cygan, P. Komosa, L. Mach, and M. Pilipczuk. Lower bounds for the parameterized complexity of minimum fill-in and other completion problems. In *SODA*, 2016.

[BDFH09] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009.

[BE00] S. P. Borgatti and M. G. Everett. Models of core/periphery structures. *Social networks*, 21(4):375–395, 2000.

[BFMY83] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

[BFPP14] I. Bliznets, F. V. Fomin, M. Pilipczuk, and M. Pilipczuk. A subexponential parameterized algorithm for proper interval completion. In *ESA*, volume 8737 of *Lecture Notes in Computer Science*, pages 173–184. Springer, 2014.

[BFPP16] I. Bliznets, F. V. Fomin, M. Pilipczuk, and M. Pilipczuk. Subexponential parameterized algorithm for interval completion. In *SODA*, 2016.

[BHPT93] Z. Blázsik, M. Hujter, A. Pluhár, and Z. Tuza. Graphs with no induced C_4 and $2K_2$. *Discrete Mathematics*, 115(1):51–55, 1993.

[BHV11] H. L. Bodlaender, P. Heggernes, and Y. Villanger. Faster parameterized algorithms for minimum fill-in. *Algorithmica*, 61(4):817–838, 2011.

[BJK14] Bodlaender, H. L., Jansen, B. M. P., and Kratsch, S. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014.

[BKH08] M. D. Barrus, M. Kumbhat, and S. G. Hartke. Graph classes characterized both by forbidden subgraphs and degree sequences. *Journal of Graph Theory*, 57(2):131–148, 2008.

[BL84] S. N. Bhatt and F. T. Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28(2):300–343, 1984.

[BLS99] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes. A Survey.* SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, USA, 1999.

[Bod98] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretial Computer Science*, 209(1-2):1–45, 1998.

[BP13] S. Bessy and A. Perez. Polynomial kernels for proper interval completion and related problems. *Information and Computation*, 231:89–108, 2013.

[Bra14] U. Brandes. Social network algorithmics. In *ISAAC*, volume 8889 of *Lecture Notes in Computer Science*, pages 2–3. Springer, 2014. Invited talk.

[BT01] V. Bouchitté and I. Todinca. Treewidth and minimum fill-in: grouping the minimal separators. *SIAM J. on Computing*, 31(1):212 – 232, 2001.

[BT02] V. Bouchitté and I. Todinca. Listing all potential maximal cliques of a graph. *Theor. Comput. Sci.*, 276(1-2):17–32, 2002.

[Cai96] L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

[Cai12] Y. Cai. *Polynomial kernelisation of $\mathcal{H}$-free edge modification problems*. PhD thesis, The Chinese University of Hong Kong, Hong Kong, 2012.

[CC99] Y. Cheng and G. M. Church. Biclustering of expression data. In *ISMB*, 1999.

[CC15] L. Cai and Y. Cai. Incompressibility of $\mathcal{H}$-free edge modification problems. *Algorithmica*, 71(3):731–757, 2015.

[CFK+15] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

[CLB81] D. Corneil, H. Lerchs, and L. Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3(3):163–174, 1981.

[CM12] J. Chen and J. Meng. A $2k$ kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211–220, 2012.

[CM14] Y. Cao and D. Marx. Chordal editing is fixed-parameter tractable. In *STACS*, volume 25, pages 214–225, 2014.

[CMNR97] K. Cirino, S. Muthukrishnan, N. Narayanaswamy, and H. Ramesh. Graph editing to bipartite interval graphs: exact and asymptotic bounds. In *FSTTCS*, pages 37–53. Springer, 1997.

[CMPP14] M. Cygan, D. Marx, M. Pilipczuk, and M. Pilipczuk. Hitting forbidden subgraphs in graphs of bounded treewidth. In *MFCS*, volume 8635 of *Lecture Notes in Computer Science*, pages 189–200. Springer, 2014.

[CPP+15] M. Cygan, M. Pilipczuk, M. Pilipczuk, E. J. van Leeuwen, and M. Wrochna. Polynomial kernelization for removing induced claws and diamonds. In *WG*, 2015. To appear.

[DDLS15] P. G. Drange, M. S. Dregi, D. Lokshtanov, and B. D. Sullivan. On the threshold of intractability. In *ESA*, volume 9294 of *Lecture Notes in Computer Science*, pages 411–423, 2015.

[DDS15] P. G. Drange, M. Dregi, and R. B. Sandeep. Compressing bounded degree graphs. In *Submitted*, 2015.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[DF13] R. G. Downey and M. R. Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.

[DFHT05] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

[DFPV14] P. G. Drange, F. V. Fomin, M. Pilipczuk, and Y. Villanger. Exploring subexponential parameterized complexity of completion problems. In *STACS*, volume 25, pages 288–299, Dagstuhl, Germany, 2014.

[DFPV15] P. G. Drange, F. V. Fomin, M. Pilipczuk, and Y. Villanger. Exploring the subexponential complexity of completion problems. *ACM Trans. Comput. Theory*, 7(4):14:1–14:38, August 2015.

[DFS99] R. G. Downey, M. R. Fellows, and U. Stege. A framework for systematically confronting computational intractability. In *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, volume 49, pages 49–99. AMS-DIMACS Proceedings Series, 1999.

[Die05] R. Diestel. *Graph theory (Graduate texts in mathematics)*. Springer Heidelberg, 2005.

[DM14a] P. Damaschke and O. Mogren. Editing simple graphs. *J. Graph Algorithms Appl.*, 18(4):557–576, 2014.

[DM14b] H. Dell and D. v. Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014.

[DP15] P. G. Drange and M. Pilipczuk. A polynomial kernel for trivially perfect editing. In *ESA*, 2015.

[DRSVS15] P. G. Drange, F. Reidl, F. Sánchez Villaamil, and S. Sikdar. Fast biclustering by dual parameterization. In *IPEC*, 2015.

[EC88] E. El-Mallah and C. Colbourn. The complexity of some edge deletion problems. *Circuits and Systems, IEEE Transactions on*, 35(3):354–362, 1988.

[Edm65] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3):449–467, 1965.

[EK10] D. Easley and J. Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010.

[FG06] J. Flum and M. Grohe. *Parameterized complexity theory*. Springer-Verlag New York Inc, 2006.

[FH76] S. Foldes and P. L. Hammer. *Split graphs*. Universität Bonn. Institut für Ökonometrie und Operations Research, 1976.

[FK10] F. V. Fomin and D. Kratsch. *Exact exponential algorithms*. Springer Science & Business Media, 2010.

[FKP+14] F. V. Fomin, S. Kratsch, M. Pilipczuk, M. Pilipczuk, and Y. Villanger. Tight bounds for parameterized complexity of cluster editing with a small number of clusters. *J. Comput. Syst. Sci.*, 80(7):1430–1447, 2014.

[FKTV08] F. V. Fomin, D. Kratsch, I. Todinca, and Y. Villanger. Exact algorithms for treewidth and minimum fill-in. *SIAM J. Comput.*, 38(3):1058–1079, 2008.

[FLMS12] F. V. Fomin, D. Lokshtanov, N. Misra, and S. Saurabh. Planar f-deletion: Approximation, kernelization and optimal FPT algorithms. In *FOCS*, page 470–479. IEEE, 2012.

[FLRS07] M. R. Fellows, M. A. Langston, F. A. Rosamond, and P. Shaw. Efficient parameterized preprocessing for cluster editing. In *FCT*, volume 4639 of *Lecture Notes in Computer Science*, pages 312–321. Springer, 2007.

[FMT09] T. Feder, H. Mannila, and E. Terzi. Approximating the minimum chain completion problem. *Inf. Process. Lett.*, 109(17):980–985, 2009.

[For09] L. Fortnow. The status of the P versus NP problem. *Commun. ACM*, 52(9):78–86, 2009.

[Fre79] L. C. Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1979.

[FS11] L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.

[FV13] F. V. Fomin and Y. Villanger. Subexponential parameterized algorithm for minimum fill-in. *SIAM J. Comput.*, 42(6):2197–2216, 2013.

[Gal67] T. Gallai. Transitiv orientierbare graphen. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(1-2):25–66, 1967.

[GGHN08] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Data reduction and exact algorithms for clique cover. *ACM Journal of Experimental Algorithmics*, 13, 2008.

[GGKS95] P. Goldberg, M. Golumbic, H. Kaplan, and R. Shamir. Four strikes against physical mapping of DNA. *Journal of Computational Biology*, 2(1):139–152, 1995.

[GHKZ08] J. Guo, F. Hüffner, C. Komusiewicz, and Y. Zhang. Improved algorithms for bicluster editing. In *TAMC*, volume 4978 of *Lecture Notes in Computer Science*, pages 445–456. Springer, 2008.

[GHPP13] S. Guillemot, F. Havet, C. Paul, and A. Perez. On the (non-)existence of polynomial kernels for P_l-free edge modification problems. *Algorithmica*, 65(4):900–926, 2013.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman, 1979.

[GKK+15] E. Ghosh, S. Kolay, M. Kumar, P. Misra, F. Panolan, A. Rai, and M. S. Ramanujan. Faster parameterized algorithms for deletion to split graphs. *Algorithmica*, 71(4):989–1006, 2015.

[GKLT09] S. Gaspers, D. Kratsch, M. Liedloff, and I. Todinca. Exponential time algorithms for the minimum dominating set problem on some graph classes. *ACM Transactions on Algorithms*, 6(1), 2009.

[Gol80] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

[Guo07] J. Guo. Problem kernels for NP-complete edge deletion problems: Split and related graphs. In *ISAAC*, volume 4835 of *Lecture Notes in Computer Science*, page 915–926. Springer, 2007.

[Heg06] P. Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.

[HIS81] P. Hammer, T. Ibaraki, and B. Simeone. Threshold sequences. *SIAM Journal on Algebraic Discrete Methods*, 2(1):39–49, 1981.

[HKMN10] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory of Computing Systems*, 47(1):196–217, 2010.

[HP10] M. Habib and C. Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

[HPTV07] P. Heggernes, C. Paul, J. A. Telle, and Y. Villanger. Interval completion with few edges. In *STOC*, pages 374–381. ACM, 2007.

[HS81] P. L. Hammer and B. Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981.

[IPZ01] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001.

[JHJJC96] Y. Jing-Ho, C. Jer-Jeong, and G. J. Chang. Quasi-threshold graphs. *Discrete Applied Mathematics*, 69(3):247–255, 1996.

[KD79] M. S. Krishnamoorthy and N. Deo. Node-deletion NP-complete problems. *SIAM Journal on Computing*, 8(4):619–625, 1979.

[KM86] M. Křivánek and J. Morávek. NP-hard problems in hierarchical-tree clustering. *Acta Informatica*, 23(3):311–323, 1986.

[Kre02] V. E. Krebs. Mapping networks of terrorist cells. *Connections*, 24(3):43–52, 2002.

[KS99] H. Kaplan and R. Shamir. Bounded degree interval sandwich problems. *Algorithmica*, 24(2):96–104, 1999.

[KS12] S. Kratsch and P. Schweitzer. Graph isomorphism for graph classes characterized by two forbidden induced subgraphs. In *WG*, volume 7551 of *Lecture Notes in Computer Science*, pages 34–45. Springer, 2012.

[KSS14] I. Kovác, I. Selecéniová, and M. Steinová. On the clique editing problem. In *MFCS*, volume 8635 of *Lecture Notes in Computer Science*, pages 469–480. Springer, 2014.

[KST99] H. Kaplan, R. Shamir, and R. E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM J. Comput.*, 28(5):1906–1922, 1999.

[KU12] C. Komusiewicz and J. Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012.

[KW13] S. Kratsch and M. Wahlström. Two edge modification problems without polynomial kernels. *Discrete Optimization*, 10(3):193–199, 2013.

[KW14] S. Kratsch and M. Wahlström. Compression via matroids: A randomized polynomial kernel for odd cycle transversal. *ACM Transactions on Algorithms*, 10(4):20:1–20:15, 2014.

[LG77] P. C. Liu and R. C. Geldmacher. On the deletion of nonplanar edges of a graph. In *SEICCGTC*, pages 727–738, 1977.

[LMS11] D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.

[Lok08] D. Lokshtanov. Wheel-free deletion is W[2]-hard. In *IWPEC*, volume 5018 of *Lecture Notes in Computer Science*, pages 141–147. Springer, 2008.

[LWG14] Y. Liu, J. Wang, and J. Guo. An overview of kernelization algorithms for graph modification problems. *Tsinghua Science and Technology*, 19(4):346–357, 2014.

[LWGC12] Y. Liu, J. Wang, J. Guo, and J. Chen. Complexity and parameterized algorithms for cograph editing. *Theoretical Computer Science*, 461:45–54, 2012.

[LY80] J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980.

[Mar94] F. Margot. Some complexity results about threshold graphs. *Discrete Applied Mathematics*, 49(1-3):299–308, 1994.

[Mar12] D. Marx. What's next? Future directions in parameterized complexity. In *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370, pages 469–496. Springer, 2012.

[MO04] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Trans. Comput. Biol. Bioinform*, 1(1):24–45, 2004.

[Moh01] B. Mohar. Face covers and the genus problem for apex graphs. *Journal of Combinatorial Theory, Series B*, 82(1):102–117, 2001.

[MP94] F. Maffray and M. Preissmann. Linear recognition of pseudo-split graphs. *Discrete Applied Mathematics*, 52(3):307–312, 1994.

[MP95] N. Mahadev and U. Peled. *Threshold graphs and related topics*, volume 56. North Holland, 1995.

[MPS13] N. Misra, F. Panolan, and S. Saurabh. Subexponential algorithm for d-cluster edge deletion: Exception or rule? In *MFCS*. Springer, 2013.

[MS99] R. M. McConnell and J. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241, 1999.

[MT03] Y. Metelsky and R. Tyshkevich. Line graphs of helly hypergraphs. *SIAM Journal on Discrete Mathematics*, 16(3):438–448, 2003.

[Nat99] A. Natanzon. *Complexity and approximation of some graph modification problems*. PhD thesis, Tel Aviv University, 1999.

[NG13] J. Nastos and Y. Gao. Familial groups in social networks. *Social Networks*, 35(3):439–450, 2013.

[Nie06] R. Niedermeier. *Invitation to fixed-parameter algorithms*. Oxford University Press, Oxford New York, 2006.

[NOdM12] J. Nešetřil and P. Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

[NSS00] A. Natanzon, R. Shamir, and R. Sharan. A polynomial approximation algorithm for the minimum fill-in problem. *SIAM Journal on Computing*, 30(4):1067–1079, October 2000.

[NSS01] A. Natanzon, R. Shamir, and R. Sharan. Complexity classification of some edge modification problems. *Discrete Applied Mathematics*, 113(1):109–128, 2001.

[Ola88] S. Olariu. Paw-free graphs. *Information Processing Letters*, 28(1):53–54, 1988.

[Räc08] H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *STOC*, pages 255–264. ACM, 2008.

[Ros72] D. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. *Graph theory and computing*, 183:217, 1972.

[RT78] D. J. Rose and R. E. Tarjan. Algorithmic aspects of vertex elimination on directed graphs. *SIAM Journal on Applied Mathematics*, 34(1):176–197, 1978.

[RTL76] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on computing*, 5(2):266–283, 1976.

[Sha02] R. Sharan. *Graph modification problems and their applications to genomic research*. PhD thesis, Tel-Aviv University, 2002.

[SS15] R. B. Sandeep and N. Sivadasan. Parameterized lower bound and improved kernel for diamond-free edge deletion. In *IPEC*, 2015.

[SST04] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1):173–182, 2004.

[Tar75] R. E. Tarjan. Graph theory and gaussian elimination. Technical Report CS-TR-75-526, Stanford University, Department of Computer Science, Stanford, CA, USA, 1975.

[TNS82] K. Takamizawa, T. Nishizeki, and N. Saito. Linear-time computability of combinatorial problems on series-parallel graphs. *Journal of the ACM*, 29(3):623–641, 1982.

[TSS05] A. Tanay, R. Sharan, and R. Shamir. Biclustering algorithms: A survey. *Handbook of Comp. Mol. Biol.*, 9(1-20):122–124, 2005.

[TY84] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.

[Ull84] J. D. Ullman. *Computational aspects of VLSI.* Computer Science Press, 1984.

[VHPT09] Y. Villanger, P. Heggernes, C. Paul, and J. A. Telle. Interval completion is fixed parameter tractable. *SIAM J. Comput.*, 38(5):2007–2020, 2009.

[WAN81] T. Watanabe, T. Ae, and A. Nakamura. On the removal of forbidden graphs by edge-deletion or by edge-contraction. *Discrete Applied Mathematics*, 3(2):151 – 153, 1981.

[Wil11] R. Williams. Guest column: a casual tour around a circuit complexity bound. *SIGACT News*, 42(3):54–76, 2011.

[WS11] D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms.* Cambridge University Press, 2011.

[Yan81a] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):77–79, 1981.

[Yan81b] M. Yannakakis. Edge-deletion problems. *SIAM Journal on Computing*, 10(2):297–309, 1981.

[Zac77] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, pages 452–473, 1977.

[Zah64] C. T. Zahn, Jr. Approximating symmetric relations by equivalence relations. *J. Soc. Indust. Appl. Math.*, 12(4):840–847, 1964.