National and Kapodistrian University of Athens
School of Science, Department of Mathematics

Obstructions and Algorithms for Graph Layout Problems

Dimitris Zoros

PhD Thesis

Supervised by

Prof. Dimitrios M. Thilikos

July, 2017

ABSTRACT

Modern Graph Theory is heavily influenced by the seminal work of N. Robertson and P. Seymour, known as Graph Minors. Through this work, a wealth of structural, combinatorial, as well as algorithmic, results has been introduced, in a series of papers having as ultimate goal to give an affirmative answer to Wagner's Conjecture. In this doctoral thesis we focus on the study of Obstruction Sets, one of the most important combinatorial concepts of this theory. In particular, we study the combinatorics and the algorithmic and computability aspects of graph obstructions and their relation to graph parameters emerging from Graph Layouts, Vertex Deletion Problems, and Graph Searching Problems. We consider several partial ordering relations on graphs such as minors, immersions, and contractions. Our study includes results on the existence and the computability of obstruction sets, combinatorial bounds on their size, and their interplay with parameterized complexity and kernelization.

ΠΕΡΙΛΗΨΗ

Η σύγχρονη Θεωρία Γραφημάτων έχει επηρεαστεί σε μεγάλο βαθμό από την δουλειά των N. Robertson και P. Seymour. Μέσα από αυτήν, πληθώρα από δομικά, συνδυαστικά, καθώς και αλγοριθμικά αποτελέσματα εισήχθηκαν, σε μία σειρά από εργασίες που είχε απώτερο στόχο την απόδειξη της εικασίας του Wagner. Σε αυτή την διδακτορική διατριβή επικεντρωνόμαστε στην μελέτη των Συνόλων Παρεμπόδισης, μία από τις σημαντικότερες συνδυαστικές έννοιες αυτής της θεωρίας. Πιο συγκεκριμένα, μελετάμε την συνδυαστική και αλγοριθμική πτυχή, καθώς και την υπολογισιμότητα, των γραφημάτων παρεμπόδισης, σε σχέση με παραμέτρους γραφημάτων που πηγάζουν από Διατάξεις σε Γραφήματα, Προβλήματα Διαγραφής Κορυφών και Προβλήματα Ανίχνευσης Γραφημάτων. Η μελέτη αυτή είναι βασισμένη σε σχέσεις μερικής διάταξης γραφημάτων, όπως τα ελάσσονα, οι εμβυθίσεις και οι συνθλίψεις. Η μελέτη μας περιλαμβάνει αποτελέσματα σχετικά με την ύπαρξη και υπολογισιμότητα των συνόλων παρεμπόδισης, συνδυαστικά φράγματα στο μέγεθος τους, καθώς και την αλληλεπίδρασή τους με την Παραμετρική Πολυπλοκότητα και την Πυρηνοποίηση.

PREFACE

Having read the abstract of this thesis I am sure you must be eager to find out what this story will involve. Unavoidably, we have to make a short intermission. I would like to grab this opportunity to show my appreciation and admiration to some people that guided me, or at least accompanied me, in my journey since I first decided that studying Mathematics is what I am "destined" to do. (This probably happened when I was in my early teens, but do not be afraid! I would not start thanking people I met in the 90's.)

When I finally had the opportunity to get in the amphitheaters and classrooms of the Department of Mathematics of the National and Kapodistrian University of Athens (NKUA), I was completely amazed by the world that unfolded right in front of my eyes. Mathematics have the ability to fascinate or to intimidate a young person. I was lucky enough not to be intimidated. I am very happy that I finally decided to give it a try, as it was an experience that changed the way I think and the way I perceive the world around me.

Many people continuously tried to persuade me to get back to my studies. My family and friends most certainly tried the most. When I finally

returned, my main motivation was Νίκη. She always supports me more than I could ever ask for, and this is why I gladly dedicate this thesis, as well as everything I ever accomplished, to her. I am more than blessed to have her in my life.

I also wish to dedicate this work to my best buddy in the whole world, little Χάρης, who made the last seven years of my life much more interesting and fun. Deep inside my heart, I hope this will be an inspiration for him to pursue a similar path. I feel I am very fortunate to have been given this opportunity and hope the same for him.

Νίκη and Χάρης make every single second of my life worth it and I am more than grateful to them for this. It would be great if I could express all my love and the appreciation I have for them in words, but I know that this cannot ever be possible (certainly not in the finite space I must operate).

I would like to sincerely thank my advisor Prof. Dimitrios M. Thilikos, who has been instructing my studies almost since the beginning. I am very glad to know him for almost ten years now and, most importantly, to consider him my friend as well as my mentor. He always helped me in any possible way, without ever waiting for something in return. This goes to show that, not only is he a pronounced scientist and a charismatic teacher, but also a great person.

Other than prof. Dimitrios M. Thilikos, the persons that were the closest to me during my studies, and always showed the utmost interest in my progress, are the other two members of my Three-Member committee: Prof. Stavros G. Kolliopoulos and Prof. Evaggelos Raptis. I should not neglect to mention at this point Prof. Lefteris M. Kirousis, member of my Seven-Member committee, who also showed equal, if not more, interest in my work. Their guidance was very important for me and, certainly, I would not have made it this far without it.

I warmly thank the other three members of my Seven-Member committee, Prof. Michael C. Dracopoulos, Prof. Ioannis Mourtos, and Prof.

Leonidas Pitsoulis who took a substantial amount of time from their busy schedules to assist me with their remarks and suggestions on this thesis. Their comments were always instructive and, undoubtedly, made this thesis as good as it would possibly be.

I was very fortunate to be given the opportunity to participate in the following research programs:

- *From Graph Theory to Matroids: Algorithmic Issues and Applications*[1], whose scientific coordinator was Prof. Leonidas Pitsoulis.

- *Inference on Markov Random Fields: Complexity and Algorithms*[2], whose scientific coordinator was Prof. Lefteris M. Kirousis.

The benefits I received from this participation were extremely important for my studies. Without trying to be melodramatic, were it not for the financial support I received through these programs, it would be almost impossible to continue my research. Therefore, I will always be indebted to the two coordinators, Prof. Lefteris M. Kirousis and Prof. Leonidas Pitsoulis.

In my research I was privileged to work with some very talented researchers, who helped me broaden my horizons. Our collaboration culminated in the publication of some interesting piece of work. It would not be an exaggeration if I were to say that I learned more from them than any textbook I ever read. I would like to kindly thank each and every one of them: Micah J Best, Dimitris Chatzidimitriou, Dr. Archontia C. Giannopoulou, Prof. Arvind Gupta, Prof. Menelaos I. Karavelas, Spyridon Maniatis, Clément Requilé, Iosif Salem, and Prof. Dimitrios M. Thilikos.

With some of the above we were really close and I feel I should additionally thank them for all the good times we had. A special “thank you”

[1] Co-funded by the Greek Ministry of Education and the European Union “Thales”’.

[2] Co-funded by the Greek Ministry of Education and the European Union “Academic and Research Excellence”.

must go to Dr. Archontia C. Giannopoulou, and Spyridon Maniatis, two of my closest and dearest friends, for their kindness and tolerance in my – rather extensive list of – flaws. I would also like to express my admiration for Dr. Archontia C. Giannopoulou, who is a brilliant scientist, and has always been one of my most reliable (and pleasant) collaborators.

I should not neglect to give credit to the Graduate Program in Logic, Algorithms and Computation ($\mu\Pi\lambda\forall$), as it was the place where I received the majority of knowledge I later used in my research. I want to publicly acknowledge the fact that Prof. Yiannis Moschovakis and Prof. Costas Dimitracopoulos were my sources of inspiration in my first "steps", together with Prof. Dimitrios M. Thilikos, and that I feel am honored to have been a student at their courses. It is very unfortunate that this program is now finishing its operation after 21 years of excellent academic offer.

Finally, I would like to give a special mention to our unprecedented, uncommon, uncustomary, unconventional, and (most certainly) unorthodox research group, Graphka, whose hideout is the room 116 of the Department of Mathematics of NKUA. Graphka, for its lucky members, is something beyond a research group, as we always feel free to express and to discuss our philosophical and political views in its premises. I hope I will never forget the great memories we created in the (purposely always hermetically) closed doors of room 116.

Dimitris Zoros
Kaisariani, 16/07/2017

CONTENTS

LIST OF FIGURES

Μὰ εὐτυχῶς ἡ ζωὴ δὲν ἀκούει ποτὲ τοὺς φρόνιμους νοικοκυραίους· γι' αὐτὸ καὶ πάει μπροστά. Γι' αὐτὸ ξεφύγαμε ἀπὸ τὸ φυτὸ καὶ πηδήξαμε στὸ ζῶο κι ἀπὸ τὸ ζῶο στὸν ἄνθρωπο· καὶ τώρα, ἀπὸ τὸν ἄνθρωπο τὸ σκλάβο, στὸν ἐλεύτερο. Ἕνας κόσμος πάλι καινούριος γεννιέται μὲ ὅλους τοὺς πόνους καὶ τὰ αἵματα τοῦ τοκετοῦ.

Νίκος Καζαντζάκης [195]

But fortunately life does not heed the sensible bourgeois mind, and that is why it can forge ahead; that is why we have surged beyond the plant to the animal, and from the animal to the human being. And now from the enslaved human being, we are evolving into the free one. A new world is being begotten again with all the blood of birth.

Nikos Kazantzakis [196]

CHAPTER 1

INTRODUCTION

1.1 Why Graphs?

A *Graph*[1] is a combinatorial "structure" that can be used to represent anything from networks to functions and from partial ordering relations to Markov chains. There are simple graphs, directed graphs, multi-graphs, hyper-graphs, finite or infinite graphs. In *Graph Theory* we can define paths, circles, trees, forests, cliques, grids or hyper-grids, among many other notions [10–15]. This diversity of concepts and "applications" has made Graph Theory very popular in almost every field of research in the,

[1] By the term "graph" (formally defined in Section 2.2) we do not mean the graphical representation of functions (which in fact can also be seen as representations of –infinite– graphs). This double use of the term may be confusing. When Graph Theory and its applications became popular in Greece, some computer scientists (mainly in the 80's) probably considered that an "hellenization" of the english word "graph" (which actually originates from the greek word «γράφημα») would be more convenient for the (finite) combinatorial use. Therefore for quite some time they called them «γράφος». Of course this is very reminiscent of some (few) of the Greek immigrants to the U.S.A. who, after their repatriation, used to call a car «κάρο» ("horse cart" in english), spontaneously applying the same principle.

so called, *Positive Sciences*, including *Physics, Biology, Chemistry,* even *Social and Information studies*. But, undoubtedly, the most important contribution of Graph Theory is in *Combinatorics*, *Discrete Mathematics* in general and in *Theory of Algorithms and Complexity*.

The theory of graphs is used in science for many years now, perhaps since the 19th century, but it was in the 60's when it finally got a place under the spotlight. The first problem that can be directly related to Graph Theory is the *Seven Bridges of Königsberg Problem*, solved by the great *Leonhard Euler* [43] (see page 24). The second, and perhaps the most famous one, is known as the *Four Color Theorem* and was posed by *Francis Guthrie* in 1852 (see page 32). In order to solve this problem, Mathematicians not only had to use their mental capability but also reach for the help of computer programs [25–29, 57]. Of course, back in the time when these problems embroiled Mathematicians, Graph Theory was not an autonomous field of Mathematics. The term *Graph* first appeared in Literature in 1878 by *James Joseph Sylvester*, in order to describe the way one can express invariants and co-variants in Algebra in a similar way as molecular diagrams in Chemistry [198]. Fifty years later, in the 30's, the first text book on Graph Theory was published, written by *Dénes Kőnig* [17]. This must be consider as the formal consolidation of this field.

Graph Theory became very popular in the 60's and 70's, after the raise of Computational Sciences. There may be a grain of irony in this, as the leading researchers in this field were, and still are, mostly related to pure[2] Mathematics (to name only a few *Paul Erdős, Alfréd Rényi, WilliamTutte, Frank Harary, Gabriel Andrew Dirac, László Lovász, Endre Szemerédi, Béla Bollobás, Neil Robertson, Paul Seymour, Noga Alon*). The reason behind this is that many problems of *Theoretical Computers Science* heavily involve graphs. Mathematicians found these problems appealing and,

[2]Here, we are forced to use the word "pure" as an adjective to Mathematics, although we believe that pure concepts can only exist in impure minds.

thus, decided first to "communicate" and then to collaborate with computer scientists in order to tackle them.

Nowadays, scientists that do research in Graph Theory come from many different disciplines of science. Some of the most common are *Mathematics*, *Computer Science*, and *Economics*. This proved to be very beneficial for the rapid development of Graph Theory, as techniques and ideas from a variety of contexts can be put to the test in order to solve a great deal of problems.

Of course, our point of view in this doctoral thesis will be the Mathematical one. To make it a little more precise, we study Graph Theory from the *Combinatorial*, *Logical*, and *Algorithmic* perspective.

This thesis studies a series of *Graph-Width Parameters*, defined using *Vertex* or *Edge Layouts* (see Chapter 4). As a starting point we will review one of the major breakthroughs in Graph Theory, namely the results in the *Graph Minors* series of papers ([65–87]) (see Section 2.6). One of the most omnipresent concepts of this theory is the notion of an *Obstruction* (or a *minimal forbidden graph*, see Section 2.4). Our main focus is the introduction and development of techniques for the characterization and the computation of *Obstruction Sets* for these layout parameters. This includes the study of *Vertex Removal* problems on width parameters (see Section 4.2.1) and the the detection of obstruction sets for variants of *Graph Searching Games* (Chapters 8 and 9), which are prominent versions of graph layout parameters.

To conclude, the two main scopes of this thesis are:

- *Graph Parameters* (With emphasis to graph layout parameters and graph searching numbers.)
- *Obstruction Sets* (Their finiteness and their computation.)

Starting from the next Section, we will – progressively – get into the particulars.

1.2 The Background

A graph G is a pair of sets (V, E), where the second set contains two-subsets of the first. Therefore, the set E indicates the elements of V that are "related" in some sense. This analogy immediately leads to a huge variety of structures and notions that can be represented by graphs. In the early years of Graph Theory this was the main motivation to consider graphs. Researchers used them to capture the abstract essence behind a certain structure or notion, and find its abstract properties.

From the Mathematician's point of view, Graph Theory is not necessarily "application driven", meaning that we study the properties of graphs, graph parameters and graph classes without considering the structures they may represent. Researchers in this field, no matter what their background is, consider graphs as their main mathematical object and focus on methods and technics introduced in the scope of this theory or, more generally, in the scope of other fields of Discrete Mathematics and Algorithms.

In this thesis, we follow the above approach to Graph Theory. The results we present may apply in a variety of contexts, but our motivation is not their applications. We examine these results mainly due to their theoretical interest (and "beauty").

Let us briefly present the basic – connected – components of this thesis.

1.2.1 Graph parameters

From Chapter 4 and on, we study (mainly) graph parameters. A *graph parameter* is a function **p** returning for each graph a nonnegative number (see Definition 4.0.1). Graph parameters usually fall into two main categories: (A) parameters that can be defined using *layouts* on graphs, i.e., orderings of vertices or edges (Section 4.1, see also [22]), and (B) *graph*

modification parameters (Section 4.2).

(A) Layout parameters

The most famous layout parameter is undoubtedly *treewidth* (Section 4.1.1) (and of course *pathwidth*, see Section 4.1.2). First defined in [33], and having more than 7 equivalent definitions [20], treewidth is a central notion of Modern Graph Theory (some – of the many – surveys on treewidth are [16, 18–21, 23]). Treewidth belongs to the family of *graph width parameters* which serve as measures of resemblance (topological or geometrical) of a graph to a particular graph family. For instance treewidth (denoted by **tw**) measures the degree in which a graph can be seen as a *tree*, i.e., it has the topological structure of a tree, and pathwidth (denoted by **pw**) as a *path*. Treewidth and pathwidth can be thought of as measures of the *global connectivity* of graphs. Another parameter in the same family is *carving-width* or *cutwidth*, **cw** (see Section 4.1.3 and Chapter 5), that measures the *global, edge-connectivity* of a graph.

The second type of layout parameters that we examine in the context of this thesis is the *Search Numbers* of *Search Game* variations (Chapters 8, 9). *Graph Searching* involves a team of mobil agents (usually thought of as searchers, pursuers or even cops), that aims at capturing a set of escaping agents (usually thought of as fugitives or robbers) that hide in some kind of network, represented by a graph. There are many different variations of this set-up, stemming from the different abilities or restrictions the two parts may have (see [95–98] for instance). The basic versions of graph searching (*Fugitive search games* to be more precise) we will focus on are *Edge search*, *Node search*, and *Mixed search*.

Given a search game variation, its search number is equal to the minimum number of "searchers" needed to guarantee the capture of the "fugitive" via a deterministic *search strategy*.

Graph searching has gained much attention in the last 30 years or so,

due to its enormous list of applications [95–98] . As many of the classic width parameters (treewidth, pathwidth, and *linearwidth* for instance) can also be defined through some variant of graph searching (Section 8.4), it provides us with an alternative – game theoretical – way to approach the properties of these parameters.

(B) Graph Modification parameters

This is a more general family of parameters. A graph modification parameter **p** measures the minimum number of times we have to apply an operation on a graph in order to obtain a new graph that has a certain property (Definition 4.2.3). The most well studied such parameters are the *minimum vertex cover* (page 67), the *minimum feedback vertex set* (page 67) and the *"planarity"* parameter (page 67). All these parameters are *vertex-deletion parameters*. We can use as modification operations *edge deletions*, *edge contractions* (and all other operations defined in Section 2.3), or even *additions of vertices or edges*. Basically, we can use any set of operations that can modify the graph. A consequence of this variety of different possibilities is that many interesting *graph problems* (*minimization problems* to be exact) emerge from these parameters.

In this thesis we focus on vertex deletion parameters because they encompass the following subcategory:

"Distance" to some bounded parameter

These parameters are defined using a "host" parameter. Let **p** be a parameter (it doesn't matter if it is a layout or a modification parameter), and r a nonnegative integer. We can define a parameter measuring the minimum number of vertex deletions one have to make in a graph G, so as the value of **p** in the new graph will become at most r (Section 4.2.1). These parameters have the potential to mix the two aforementioned categories

together and create many interesting problems. Their study is one of the "collateral" targets of this thesis.

1.2.2 Partial ordering relations on graphs

Graph modification can also be used to define *partial ordering relations on graphs*. The most known are the *subgraph* relation, where a graph H is a subgraph of G if it can be obtained after the removal of some edges and/or vertices of G, and the *minor* relation, where H is a minor of G, if it can be obtained from a subgraph of G after the *contractions* of some edges (Section 2.3). Two equally interesting relations are the *(weak) immersion* relation where H is obtained form a subgraph of G after some *edge lifts*, and the *contraction* relation where H is obtained after the contraction of some edges of G (Section 2.3).

For any given graph class $\mathcal{C}$ and partial ordering relation $\preceq$, there are two very important questions emerging:

(A) *Is $\mathcal{C}$* closed under $\preceq$ *(or $\preceq$-closed), i.e., does it hold that for any given graph $G \in \mathcal{C}$, for every graph $H \preceq G$, $H \in \mathcal{C}$?*

(B) *(If $\mathcal{C}$ is a infinite class) is $\mathcal{C}$ well-quasi ordered with respect of $\preceq$, i.e., for every infinite subset of $\mathcal{C}$ do there exist two graphs, say H and G, such that $H \preceq G$?*

We are interested to know the answer to these questions for classes containing graphs whose value of the parameter we examine is bounded by some positive integer. Let $\mathcal{G}$ be the class of all graphs and $\mathcal{G}[\mathbf{p}, k]$ the class of all graphs G such that $\mathbf{p}(G) \leq k$.

The first question is typically easy to be answered. The second is one of the most difficult questions in Graph Theory. For instance, the proof that $\mathcal{G}$ is well-quasi ordered under the minor relation, known as the *Robertson–Seymour Theorem* or the *Graph Minors Theorem*, took Neil

Robertson and Paul Seymour almost 30 years to be written, and extends to 23 papers ([65–87], see also [64]).

From the *Graph minors series* we also know that $\mathcal{G}$ is well-quasi ordered under the (weak) immersion relation. Unfortunately, this is not true for the other two relations, the subgraph and the contraction relations.

The importance of questions (A) and (B) above, is discussed in the next Section.

1.2.3 Obstructions

Given a graph class $\mathcal{C}$ that is $\preceq$-closed, we denote by $\mathsf{obs}_{\preceq}(\mathcal{C})$ the set of minimal graphs, with respect of $\preceq$, not belonging to $\mathcal{C}$. This set is called the *obstruction set* of $\mathcal{C}$ with respect of $\preceq$ (Section 2.4). Observe that for any graph G for which there exists a graph $O \in \mathsf{obs}_{\preceq}(\mathcal{C})$, such that $O \preceq G$, we can immediately conclude that $G \notin \mathcal{C}$, as, if G were in $\mathcal{C}$, then also O would be in $\mathcal{C}$. This set, in a sense can characterize $\mathcal{C}$ by forbidding some graphs.

As $\mathcal{G}$ is well-quasi ordered under the minor ($\leq_{\mathrm{m}}$) and the (weak) immersion ($\leq_{\mathrm{im}}$) relation, this yields that the sets $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathcal{C})$ and $\mathsf{obs}_{\leq_{\mathrm{im}}}(\mathcal{C})$ will be finite for every minor or immersion-closed graph class $\mathcal{C}$. Thus, we have a finite *Forbidden graph characterization* or *Kuratowski characterization*[3] for these classes. This characterization is completely "independent" from every combinatorial, topological or geometrical properties $\mathcal{C}$ may have, and therefore, can be (almost[4]) trivially used to device algorithms that check whether a graph belongs to $\mathcal{C}$ or not. The only thing we have to check is whether the graph contains some graph of the obstruction set as minor (or immersion).

[3] Named after Kazimierz Kuratowski who gave the first such characterization, i.e., the one characterizing planar graphs as those that do not contain K_5 or $K_{3,3}$ as topological-minors [51].

[4] We also need a minor checking (Theorem 2.6.2) and an immersion checking algorithm (Theorem 2.6.3).

The bad news are that the proof of the Robertson–Seymour Theorem, as well as its immersion counterpart, are not, and cannot be, *constructive* [88]. This roughly means that we know the obstruction sets are finite but there is no way, using the *Graph Minors*, to design an algorithm that finds them. Unavoidable, we have to improvise for each and every graph class, or to find massive classification technique in order to enlarge the computability horizon of this theory. The "central" target in this thesis is to contribute to this direction.

1.3 The Foreground

Here we study the existence of finite Kuratowski characterizations for classes of graphs with bounded width parameters. As far as the minor or the immersion relation is concerned, the question is not *if* such characterization exists, but whether it can be *computed*, and which are the necessary requirements for this to happen.

We present the results in this thesis, starting with the minor ordering relation. Then, we move on to the immersion and contraction relations.

1.3.1 Monotone kernels

Much work has been devoted to finding the necessary conditions for the computability of obstruction sets for the minor relation in the last decades [77,80,85–87,92,123,125]. In Chapter 7 we consider minor-closed graph parameters that meet two additional conditions:

- they are *Protrusion decomposable*, and
- they have *Finite Integer Index* (FII).

(We formally define this properties in Sections 7.1 and 7.2.)

One of the results presented in this thesis is that, when a minor-closed parameter **p** has these properties and its value drops by a constant factor if we do some local transformation to a graph, then the size of the graphs in $\mathsf{obs}_{\leq_m}(\mathcal{G}[\mathbf{p}, k])$ is *linearly* bounded by k. Moreover, when **p** is computable and the FII property is constructive (for more on this "delicate" issue see Section 7.7), the set $\mathsf{obs}_{\leq_m}(\mathcal{G}[\mathbf{p}, k])$ becomes computable.

The central theorem of Chapter 7 can be stated as follows:

Theorem 7.3.1. *For every graph parameter* **p** *that has* FII*, is computable, protrusion decomposable, and minor-closed, there is a constant* $c_{\mathbf{p}}$ *and a polynomial algorithm that given a graph* G*, outputs a graph* G' *such that*

1. $G' \leq_m G$,
2. $\mathbf{p}(G') = \mathbf{p}(G)$, *and*
3. *the size of* G' *is at most* $c_{\mathbf{p}} \cdot \mathbf{p}(G)$.

Using this theorem we can draw some very interesting conclusions about the existence of *linear kernels* for *Graph optimization problems*. A *kernelization algorithm* for a parameterized graph problem[5] – or, simply, a *kernel* – is a polynomial-time algorithm that transforms every instance (G, k) of the problem to an equivalent one (G', k'), where the size of G' and the integer k' depend exclusively on the parameter k (Definition 3.3.1). Ideally this dependency is linear (and therefore we have a *linear kernel*). Theorem 7.3.1 proves the existence of linear kernels for optimization graph problems where **p** is the function returning the "size" of the *optimal solution* of an instance (see Definition 7.4.1). Properties *1.* and *2.* of Theorem 7.3.1 give two additional properties to these kernels:

[5] A parameterized problem can be seen as a subset of $\Sigma^* \times \mathbb{N}$ where we have instances of the form (x, k), i.e., a word (an instance in the "classic" sense) and an integer k, which is the *parameter* of the problem. In parameterized graph problems x encodes a graph. Section 3.2 contains all the details.

- They are *minor monotone*, i.e., $G' \leq_{\mathrm{m}} G$.

- They are *parameter invariant*, i.e., $k' = k$.

Such kernels present independent interest as, apart from their implication to obstruction sets (Section 7.4.4), they can also accelerate known approximation schemes (Section 7.4.5). We will not present the consequences of the existence of such kernels here (we do so in Chapter 7), as we do not want to overwhelm the reader with definitions right form the introduction. Here we will only discuss the following result.

"Distance" to bounded parameter

Let as use as "host" the parameter **p**, and define the family of vertex removal parameters $(\mathbf{p}, r)$-dist, $r \in \mathbb{N}$, where, for very graph $G = (V, E)$:

$$(\mathbf{p}, r)\text{-dist}(G) = \min\{k \mid \exists S \subseteq V \text{such that } |S| \leq k \text{ and } \mathbf{p}(G \setminus S) \leq r\}$$

(Definiton 4.2.4)

Combining the fact that the $\mathcal{F}$-Covering minimization problems (Section 4.2.1 and SubSection 7.4.2) admit linear, minor monotone and parameter invariant kernels, with properties of these problems recently proved [174,175], we can show that the obstructions of the set $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathcal{G}[(\mathbf{p}, r)\text{-dist}, k])$, r, $k \in \mathbb{N}$, that are *H-topological-minor free*[6] for some graph H, can be computed, when **p** is a minor-closed, computable parameter, such that

- for a non-connected graph the value of **p** is equal to the maximum value of its connected components (Definition 7.4.10), and

- its value in *grids* is not bounded (Definition 4.2.6).

[6] A graph G is H-topological-minor free if H cannot be obtained from G after a series of vertex and/or edge deletion and vertex *dissolution*.

(Theorem 7.4.3)

A large family of parameters having the properties needed for Theorem 7.4.3 described above, is the graph searching numbers. For example, let us examine the search numbers of the three aforementioned variations: **es** for the Edge search, **ns** for the Node search, and **ms** for the Mixed search. We prove that, for every $r, k \in \mathbb{N}$, we can compute the obstructions of the sets $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathcal{G}[\bullet, k)]$, where $\bullet \in \{(\mathbf{ns}, r)\text{-dist}, (\mathbf{es}, r)\text{-dist}, (\mathbf{ms}, r)\text{-dist}\}$, that are H-topological minor-free, for some graph H (Section 8.5.1).

1.3.2 Obstructions for unions of classes

As we said in the beginning of the previous section, obstructions for the minor relation have been extensively studied. A quite interesting result is the fact that if we are given the obstruction sets for two minor-closed graph classes, say $\mathcal{C}_1$ and $\mathcal{C}_2$ then we can compute the obstruction set for the class $\mathcal{C}_1 \cup \mathcal{C}_2$ [123, 125]. This implies that the constructibility of the Graph Minors Theory is "closed" with respect to the union operation.

In Chapter 6 we deal with the counterpart of this problem for the immersion relation. We build on the machinery introduced by Isolde Adler, Martin Grohe and Stephan Kreutzer in [123], for computing minor obstruction sets, and prove the existence of an algorithm computing the immersion obstruction set of a graph class $\mathcal{C}$ with the following properties:

- $\mathcal{C}$ is immersion-closed (of course),
- it has an MSO-description[7], and
- an upper bound on the treewidth of the subgraph-minimal graphs containing obstructions of $\mathcal{C}$ is known.

[7] I.e., There exist a *Monadic Second-Order logic* formula $\phi_{\mathcal{C}}$ such that $G \in \mathcal{C}$ if and only if $\phi_{\mathcal{C}}$ is true for G (Definition 2.7.5).

(Corollary 6.2.2)

Taking this last property into consideration, we prove that there exists a uniform upper bound on the treewidth of the subgraph-minimal graphs that do not belong to the union of two immersion-closed graph classes, whose immersion obstruction sets are known (Lemma 6.3.3). This result makes use of an extension of the *Unique Linkage Theorem* of Kawarabayashi and Wollan [89].

Combining this bound with the aforementioned algorithm, we can show that the immersion obstruction set of the union of two graph classes can be computed, provided that the two immersion obstruction sets are known (and given) to us.

1.3.3 d-cutwidth

Our motivation to look into cutwidth under the immersion relation prism, is the fact that it is not a minor-closed parameter (e.g., [22]).

We noticed that cutwidth can be seen as the "unidimensional" version of a "multidimensional" parameter, called d-cutwidth (we denote it by $\mathbf{cw}_d$, where d stands for the dimension of the space we work on) (see Definitions 5.1.1 and 5.1.2 and Theorem 5.1.1). 1-cutwidth (i.e., cutwidth) is defined as the minimum over all possible vertex layouts, of the maximum cost a layout may have (Definition 4.1.6). The cost of a layout is defined as the number of edges crossing a cutting point in this layout (we will refer to this point as the "cut"). In d-cutwidth we extend the notion of vertex layouts, which can be seen as embeddings in a straight line segment, to embeddings in $\mathbb{R}^d$ (see Section 5.1). Moreover, we define the "cut" as the interSection of *hyperplanes* of $\mathbb{R}^d$ with this embedding.

We have to postpone the formal definitions until Chapter 5, as they probably are out of the scope of this introduction as well. In Chapter 5 we will prove some of the properties d-cutwith has (we gathered them in Theorem 5.4.1), and discuss the difficulties posed by the fact that embed-

dings in $\mathbb{R}^d$ contain points that may have some irrational coordinates (see Section 5.3). Hence, the discretization of these embeddings is mandatory if we want to examine the *Computational Complexity* of finding the d-cutwidth of a graph.

To conclude that, as we indeed proved, this parameter is immersion-closed, therefore the classes $\mathcal{G}[\mathbf{cw}_d, k]$, for every $d, k \in \mathbb{N}$, admit a finite forbidden immersions characterization. Yet, we do not have any idea of devising an algorithm producing these characterizations. Notice that if we find such an algorithm, then there is no need to discretize embeddings to compute the d-cutwidth of a graph. A consequence of our results is that checking whether the d-cutwidth of a graph is at most k can be done (non-constructively[8]) in linear time (Proposition 5.3.1).

1.3.4 Connected searching

The case where the question of the existence of *finite* Kuratowski characterization is meaningful is when we consider the contraction relation. As we already mentioned, $\mathcal{G}$ is not well-quasi ordered under the contraction relation (an infinite *anti-chain* for this relation is depicted in Figure 2.5), which means that a contraction-closed graph class $\mathcal{C}$ may not (and typically do not[9]) have a finite such characterization. Then...

> *Why should we ever bother trying to find obstruction sets of contraction-closed classes?*

Let us get back to graph searching numbers. We are particularly interested in the *Mixed search* variant where the part of the graph that is

[8] As the computation of the immersion obstruction sets for $\mathbf{cw}_d$ – in general – is not constructive.

[9] Other than the contraction obstruction sets in [1, 2] the only (non-trivial) example in literature of finite such set is the set in [128], characterizing planar graphs. In [99, 132] we can see examples of infinite contraction obstruction sets.

restricted for the fugitive is connected (see Section 8.3). In this way, searchers can communicate safely with each other through this area and coordinate their efforts. This variant, known as *Connected search*, has recently gained much attention [1, 2, 100, 107].

The search number of this game, is neither minor- nor immersion-closed, because the deletion of edges may "disconnect" the graph. In this case the *connected search number* of the new graph will be infinite. Interestingly enough, it is contraction-closed. This motivated us to investigate the possibility of finding forbidden contractions characterizations for the classes of graphs of bounded connected search numbers.

In Chapter 9 we prove that the set of forbidden contractions, characterizing the class $\mathcal{C}$ of graphs that can be searched using at most two searchers[10], using only *connected search strategies*, is finite and consists of **177** graphs. Moreover, we managed to identify every graph in this set. We will describe these graphs, alongside with the restrictions they impose on $\mathcal{C}$. One of these restrictions is the direction the searchers must use to traverse the graph. We will show that this direction plays a very important role for the success of their search strategy towards capturing the fugitive. As a matter of fact, the majority of forbidden graphs in this set are there to fix the course along an imaginary axis of the graph the searchers must follow to achieve their goal.

1.4 Some Remarks on the Structure

In Chapter 2 we introduce the main notations that concern anything from sets, function and logic formulas to graphs. After this, we present the story behind the *Robertson–Seymour Theorem*. This story starts with the first Kuratowski characterization, namely the chatarcterization of planar

[10] Two may be too little, but it seems really hard to extend our results for $k \geq 3$. The reason is that, even if we know that the obstruction set for some $k \in \mathbb{N}$ is indeed finite, this set would contain more than $2^{2^{\Omega(k)}}$ graphs (see Section 9.3).

graphs as the graphs that are K_5 and $K_{3,3}$ topological-minor free, and accompanies Graph Theory ever since.

The problems we consider solving algorithmically in the context of this thesis belong to the class NP. In Chapter 3 we will look deeper into the structural properties of the instances that such problems may have. This motivated the definition of multivariable measures of complexity. To be more precise, we investigate these problems from the *Parameterized Complexity* [139–143] point of view. In this complexity theory we distinguish a structural measure of the input, other than its size. This measure is called *the parameter* of the problem and, accordingly, the problem is a *parameterized problem*. The challenge is to design algorithms that take into consideration that, in most practical cases, the parameter is small (much smaller than the size of the input) and thus we may tolerate some super-polynomial contribution of the parameter in the running time of the algorithms. This parameterized viewpoint reveals a more accurate picture on the complexity status of the problems.

In this chapter we also present the concept of *Kernelization algorithms* and define graph optimization problems.

Chapter 4 consists of the definitions of the graph parameters we consider. As the searching numbers have independent interest, we chose not to give their definitions in this chapter, but present them in a separate chapter instead (see Chapter 8).

We also choose to do the same with d-cutwidth. We moved its definition to Chapter 5, where we will have room to set the necessary notation, as well as to discuss its properties in detail.

The first[11] computability result for obstruction sets we present, concerns the union of immersion-closed graph classes, where their respective

[11] To be precise, this will be the second such result. In Section 4.2.1, where we discuss the "distance" to bounded parameters, we present some preliminary and already known results.

obstruction sets are known. It is presented in Chapter 6. Although in the introduction we chose to look first at obstruction sets for the minor relation, we feel it would be better to familiarize the reader of this thesis with the question of computation of these sets, using the more instructive proofs of Chapter 6, rather than the technical proofs of Chapter 7.

Chapter 7 may contain a wealth of notions and technics, but, on the downside, it may be a little difficult to follow. We broke the proof of Theorem 7.3.1 into small parts, and give some intuition whenever this was possible. Here, we will further discuss optimization problems and kernels.

The last two chapters of this thesis are devoted to Graph Searching. Chapter 8 starts with a short historical retrospection of graph searching games, more precisely *Fugitive search games*, and the definition of their basic variations. Then we present two important notions: *Monotonicity* and *Connectivity*, and discuss the connection between search numbers and – the already defined in Chapter 4 – layout parameters.

In Chapter 9 we present our work on connected graph searching. Again, the proofs presented here contain a lot of details and case analysis, but much effort has been made to guide the reader through this proof.

We close this – introductory – chapter by presenting a list of publications that are (or are not) based on parts of this thesis. Then we get into business!

1.5 The Papers

The part of our research presented in this thesis was published in the following papers (chronologically ordered):

[4] Archontia C. Giannopoulou, Iosif Salem, and Dimitris Zoros, *Effective Computation of Immersion Obstructions for Unions of Graph Classes*, Journal of Computer and System Sciences, Volume 80, Issue 1, pp. 207–216, 2014.

This publication is based on parts of Chapters 2, 4, and 6.

[5] Archontia C. Giannopoulou, Iosif Salem, and Dimitris Zoros, *Effective Computation of Immersion Obstructions for Unions of Graph Classes*, Scandinavian Workshop on Algorithm Theory (SWAT 2012), pp. 165-176, 2012.

This paper is a preliminary version of [4].

[1] Micah J Best, Arvind Gupta, Dimitrios M. Thilikos, and Dimitris Zoros, *Contraction Obstructions for Connected Graph Searching*, Discrete Applied Mathematics, Volume 209, pp. 27–47, 2016.

This publication is based on parts of Chapters 2, 8, and 9.

[2] Micah J Best, Arvind Gupta, Dimitrios M. Thilikos, and Dimitris Zoros, *Contraction Obstructions for Connected Graph Searching*, 9th International Colloquium on Graph Theory and Combinatorics (ICGT 2014), 2014.

This paper is a preliminary version of [1].

[6] Menelaos I. Karavelas, Spyridon Maniatis, Dimitrios M. Thilikos, and Dimitris Zoros, *Geometric Extensions of Cutwidth*

in any Dimension, 9th International Colloquium on Graph Theory and Combinatorics (ICGT 2014), 2014.

This paper is based on parts of Chapters 2, 3, 4, and 5.

[3] Dimitris Chatzidimitriou, Dimitrios M. Thilikos, and Dimitris Zoros, *Parameter Invariant, Minor-monotone Kernels*, Unpublished Manuscript, Submitted to: 12th International Symposium on Parameterized and Exact Computation (IPEC 2017), 2017.

This paper is based on parts of Chapters 2, 3, 4, and 7.

1.6 Not Included in This Thesis

As the scope of this thesis contains mainly two notions, *Obstruction Sets* and *Graph Layouts*, the following paper seems to be out of scope. Nevertheless, we should not avoid mentioning it.

[7] Dimitris Chatzidimitriou, Archontia C. Giannopoulou, Spyridon Maniatis, Clément Requilé, Dimitrios M. Thilikos, and Dimitris Zoros, *Fixed Parameter Algorithms for Completion Problems on Planar Graphs*, 41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016), 2016.

There were two preliminary versions of this paper:

[8] Dimitris Chatzidimitriou, Archontia C. Giannopoulou, Spyridon Maniatis, Clément Requilé, Dimitrios M. Thilikos, and Dimitris Zoros, *Fixed Parameter Algorithms for Completion Problems on Planar Graphs*, Algorithmic Graph Theory on the Adriatic Coast (AGTAC 2015), 2015.

[9] Dimitris Chatzidimitriou, Archontia C. Giannopoulou, Clément Requilé, Dimitrios M. Thilikos, and Dimitris Zoros, *A Fixed Parameter Algorithm for Plane Subgraph Completion*, 13th Cologne-Twente Workshop on Graphs & Combinatorial Optimization (CTW 2015), 2015.

CHAPTER 2

BASIC DEFINITIONS

In this Chapter we will give the basic definitions and fix the notation we plan to use throughout this thesis. Although we are going to present the basic notations in a casual way, we will not compromise the formality needed for this occasion. Before we get into graphs, we have to talk about sets and functions.

2.1 Sets And Functions

We use the logic symbols $\wedge, \vee, \neg, \rightarrow, \Rightarrow, \Leftrightarrow, \in, \setminus, \cup$ and $\cap$ in the standard way[1], and, occasionally, use $\mathbb{N}$ for the set of natural numbers ($\mathbb{N}^+$ for the set of natural numbers), $\mathbb{Z}$ for the set of integers ($\mathbb{Z}^+$ for the set of positive integers) and $\mathbb{R}$ for the set of real numbers ($\mathbb{R}^+$ for the set of positive real numbers).

To keep the notation as short as possible we will write $[n]$ instead of $\{1, \ldots, n\}$, $n \in \mathbb{N}$.

[1] We may slightly deviate from the "standard" and write $\cup\{F_1, \ldots, F_n\}$ and $\cap\{F_1, \ldots, F_n\}$ instead of $F_1 \cup \cdots \cup F_n$ and $F_1 \cap \cdots \cap F_n$.

Given a set S, we denote by 2^S the set of all subsets of S and by $\binom{S}{2}$ the set of all subsets of S with cardinality 2.

Given a function $f : A \to B$ and a set S, we define $f|_S = \{(x, f(x)) \mid x \in S \cap A\}$ and $f \setminus S = \{(x, f(x)) \mid x \in A \setminus S\}$. Moreover, we always assume that a function $\sigma : A \to B$ is also defined on 2^A so that for $S \subseteq A$, $\sigma(S) = \{\sigma(x) \mid x \in S\}$. We denote by $\emptyset$ the empty set $\{\}$ and by $\varnothing$ the empty function, i.e., $\varnothing : \emptyset \to \emptyset$.

2.2 Graphs

A *graph* is a pair $G = (V, E)$ of sets where E consists of two-subsets of V. The set V is the *vertex set* and the set E the *edge set* of G. We will refer to the elements of these two sets as *vertices* and *edges* respectively. Throughout this thesis we will depict the vertices of a graph as black dots (in some cases we will use additional shapes to distinguish some special vertices) and the edges as line segment (usually straight) connecting their *endpoints*, i.e., the two elements of V that constitute the edge. Later, in Chapter 5, we will discuss about embeddings of graphs in *Euclidean spaces* and somehow justify why we chose this way to depict graphs.

Given a graph G, when we do not explicitly state the names of its vertex and edge set, we will denote them as $V(G)$ and $E(G)$ respectively. The two basic "measures" of a graph is the number of vertices $n(G) = |V(G)|$ (we may sometimes use $|G|$ instead, or just n when the context makes clear that it represents $n(G)$) and the number of edges $m(G) = |E(G)|$. In the following Chapters we will see numerous other measures of a graph, measuring – loosely speaking – its "width".

If $S \subseteq V(G)$ we call graph $G[S] = (S, \big\{\{u, v\} \in E(G) \mid u, v \in S\big\})$ the *subgraph of G induced by S*. Accordingly, given a set $F \subseteq E(G)$ we call graph $G[F] = (\bigcup_{e \in F} e, F)$ the *subgraph of G induced by F* and we denote by $V(F)$ the set of vertices in $G[F]$. We denote by $G \setminus S$ the graph $G[V(G) \setminus S]$.

Let $u \in V(G)$ be a vertex of a graph G. We adapt the standard notations for the *(open) neighbourhood* and the *degree* of u, that is the set of all vertices connected with u by an edge, denoted by $N_G(u)$, and the cardinality of this set, denoted by $\deg_G(u)$. Moreover, for $S \subseteq V(G)$ we define $N_G(S) = \cup_{u \in S} N_G(u)$. The *closed neighbourhood* of S in G is $N_G[S] = S \cup N_G(S)$. We also define $\partial_G(S)$ to be the set of all vertices of S that are incident to edges not in $G[S]$.

A *path* of *length* $k \geq 1$ is a graph $P = (V, E)$ where $V = \{u_0, u_1, \ldots, u_k\}$ and $E = \{\{u_0, u_1\}, \{u_1, u_2\}, \ldots, \{u_{k-1}, u_k\}\}$. In this case we say that u_0 and u_k are the *ends* of P or, in other words, that Pconnects u_0 and u_k (we may refer to P as a (u_0, u_k)*-path*). All other vertices of P are *internal vertices*.Two paths are *edge-disjoint* if they do not share common edges and *vertex-disjoint* if they do not share common vertices.

A closed path, i.e., the graph $C = (V, E)$ with $V = \{u_1, u_2, \ldots, u_k\}$ and $E = \{\{u_1, u_2\}, \{u_2, u_3\}, \ldots, \{u_{k-1}, u_k\}, \{u_k, u_1\}\}$, is also called a *cycle* of *length* $k \geq 1$.

A graph G is *connected* if for every two vertices $u, v \in V(G)$, G contains a path connecting them.

If G is not connected and $V_1, \ldots, V_k$ are the maximal, under the subset relation, vertex sets that induce connected graphs, then we define $G[V_i]$, $1 \leq i \leq k$, to be the *connected components* of G.

Let G be a graph with at least three vertices. G is *2-connected* if for every two vertices $u, v \in V(G)$, G contains two paths with u and v as ends that meet only at their ends (i.e., apart from u, v they are vertex-disjoint).

If G is not 2-connected, then the maximal vertex sets that induce 2-connected graphs are the *2-connected components*, or *blocks*, of G.

A vertex of a graph is *isolated* if it has degree 0 and *pendant* if it has degree at most 1. Accordingly, an edge e is *pendant* if one of its endpoints is pendant. If both endpoints of e are pendant, then we say that e is an *isolated edge*.

Let u and v be two vertices of G. The *distance of* u *to* v is the minimum

length of a path in G, having u and v as ends.

If a graph G does not contain a cycle, of any length, then it is called a *forest*. In addition, if G is connected then it is called a *tree*.

Let T be a tree. The *leaves* of T are its vertices that have degree at most 1. The set of leafs of T is denoted by $\mathsf{Leaf}(T)$.

Given a tree T and two distinct vertices a, b of $V(T)$ we denote by aTb the – unique – path in T connecting a and b.

A *clique* of $k \geq 1$ vertices, or k-clique, is the graph $K_k = \big(\{u_1, \ldots, u_k\}, \big\{\{u_i, u_j\} \mid 1 \leq i < j \leq k\big\}\big)$. For two disjoint sets A, B, of cardinality k, l respectively, we define the graph $K_{k,l} = (A \cup B, \big\{\{u, v\} \mid u \in A \text{ and } v \in B\big\})$.

The *line graph* of a graph G, denoted by $L(G)$, is the graph $(E(G), X)$, where $X = \{\{e_1, e_2\} \subseteq E(G) \mid e_1 \cap e_2 \neq \emptyset \wedge e_1 \neq e_2\}$.

Let F be a set of edges not sharing common endpoints with each other. If $\cup_{e \in F} e = V(G)$, then F is a *perfect matching* of the vertices of G.

Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ we define their *union* to be the graph $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. When V_1 and V_2 are disjoint, we refer to this union as the *disjoint union* of G_1 and G_2 (we denote this fact by $G_1 + G_2$). The *lexicographic product* $G_1 \times G_2$ is the graph with $V(G_1 \times G_2) = V(G_1) \times V(G_2)$ and $E(G_1 \times G_2) = \{\{(x, y), (x', y')\} \mid (\{x, x'\} \in E(G_1)) \vee (x = x' \wedge \{y, y'\} \in E(G_2))\}$.

Given a $k \in \mathbb{N}^+$, we define the $(k \times k)$-*grid* as the graph $P_{k-1} \times P_{k-1}$ and we denote it by $\boxplus_k$.

Two graphs, say G and G' are *isomorphic* if there exists a bijection $\phi : V(G) \to V(G')$ such that $\{u, v\} \in E(G) \Leftrightarrow \{\phi(u), \phi(v)\} \in E(G')$ for every u, v in $V(G)$. In this case, function ϕ is an *isomorphism*.

This concludes the basic definitions about graphs. We are now able to define some interesting problems about them.

Historically the fist problem associated with Graph Theory is the *Seven Bridges of Königsberg Problem* [199]:

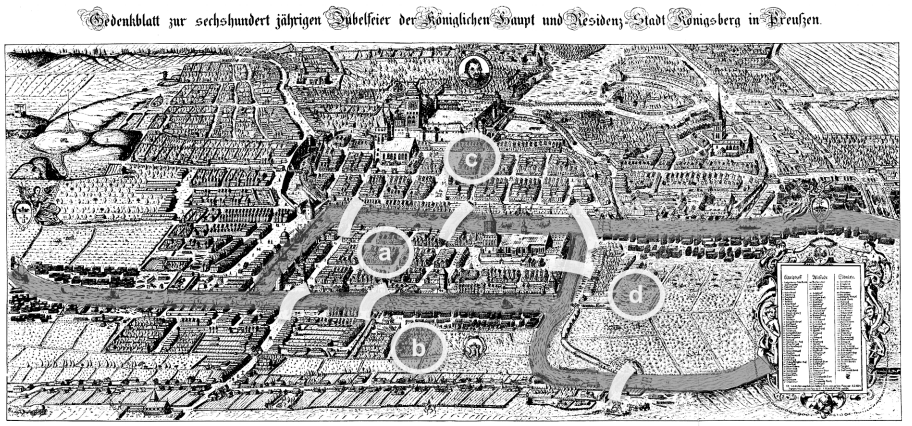


Figure 2.1: The seven bridges of Königsberg.

Can all seven bridges of Königsberg (Figure 2.1) be traversed in a single trip without doubling back, with the additional requirement that the trip ends in the same place it began?

In terms of Graph Theory the question is whether the underlying graph (Figure 2.2), where bridges correspond to edges, has an *Eulerian cycle*. In 1736 Leonhard Euler gave an answer to this problem, hence, we have the first Theorem of Graph Theory appearing in literature:

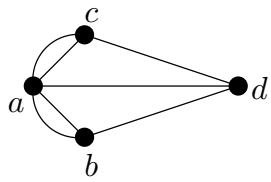


Figure 2.2: The graph formed from the seven bridges of Königsberg.

Theorem 2.2.1 (Euler, 1736 [43])**.** *A connected graph has an eulerian cycle if and only if every vertex has even degree.*

Königsberg has lost two of its seven bridges during World War II and later, in 1945, was named Kaliningrad after Mikhail Kalinin (Михаил Иванович Калинин)[2]. Sadly, World War II not only changed the name of the problem, but also changed its solution, as – it may still not admit an Eulerian cycle – it now admits an *Eulerian path* [194].

In the following Chapters we will discuss many more problems in the field of Graph Theory, and see their solutions and the techniques involved in them. Before we proceed in this endeavour we have to establish our notation and get a glimpse of some central results.

2.3 Contractions, Minors and Immersions

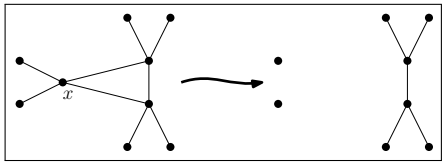
There are many different operations locally transforming a graph to a new graph, the most common of which are the following:

$\setminus u$ **:** The *deletion* of a vertex $u \in V(G)$ tranforms G to graph $G \setminus u = (V(G) \setminus \{u\}, \{e \in E(G) \mid u \notin e\})$ (see Figure 2.3a).

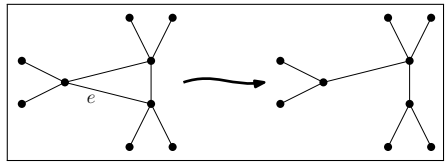
$\setminus e$ **:** The *deletion* of an edge $e \in E(G)$ transforms G to graph $G \setminus e = (V(G), E(G) \setminus \{e\})$ (see Figure 2.3b).

$/e$ **:** The *edge contraction* of $e = \{u, v\}$ (or just the *contraction* of $\{u, v\}$) is the operation that deletes this edge, adds a new vertex x_{uv} and

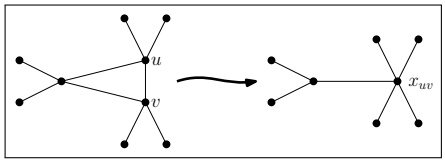
[2] But why Stalin named Königsberg, the birth place of the great Immanuel Kant, after Kalinin, a man with no real power or influence? According to Milan Kundera's book *The Festival of Insignificance* [197] Kalinin had a bladder problem that forced him to urinate at very frequent intervals. Stalin, being aware of this fact, would intentionally slow down his story telling. Out of respect, poor Kalinin would remain sitting during the whole story, thereby intensifying his bladder problem. It was only when Kalinin would finally give up, shaming himself in front of the other Congress members, that Stalin would bring his anecdotes to a close. This undoubtedly proves Kalinin's devotion to his leader, who later acknowledging this fact named Königsberg after him. This story may not seem legit but, at least to me, is very amusing. What make things even more particular is that Königsberg is still called Kaliningrad, but, for instance, Leningrad changed back to Saint Petersburg in 1991 and Stalingrad was named Volgograd in 1962.



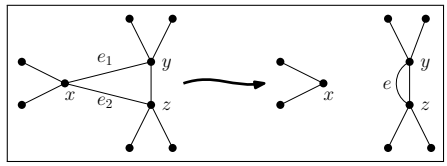
(a) The deletion of x.



(b) The deletion of e.



(c) The contraction of $\{u, v\}$ to x_{uv}.



(d) The lift of e_1 and e_2 to e.

connects this vertex to all the neighbours of u and v (if some multiple edges are created we delete them). G/e is the graph obtained (see Figure 2.3c). Moreover, if an endpoint of e, say u, has degree 2 then the contraction of e is also called *dissolution* of vertex u. We denote by G/u the graph obtained.

Lifts: The *lift* of two edges $e_1 = \{x, y\}$ and $e_2 = \{x, z\}$ to an edge e is the operation of removing e_1 and e_2 from G and then adding the edge $e = \{y, z\}$ in the resulting graph. Notice that if $\{y, z\}$was already present in G, then this operation creates a multiple edge between y and z (this is the only exception throughout this thesis where multiple edges are allowed, see Figure 2.3d for an example).

Based on these operations we can define various relations on graphs (consequently, these relations can define partial orderings on graphs, some of which we discuss in the next Section). In a nutshell, let H and G be two graphs, then:

- H is a *subgraph* of G if it can be obtained after a series of vertex and edge deletions on G. We denote it by $H \leq G$.

- H is a *spanning subgraph* of G if it can be obtained after a series of edge deletions on G. We denote it by $H \leq_{\text{sp}} G$.

- H is an *induced subgraph* of G if it can be obtained after a series of vertex deletions on G. We denote it by $H \leq_{\text{in}} G$.

- H is a *contraction* of G if it can be obtained after a series of edge contractions on G. We denote it by $H \leq_{\text{c}} G$.

- H is a *topological-minor* of G if it can be obtained after a series of vertex deletions, edge deletions and vertex dissolutions on G. We denote it by $H \leq_{\text{tp}} G$.

- H is a *minor* of G if it can be obtained after a series of vertex deletions, edge deletions and edge contractions on G. We denote it by $H \leq_{\text{m}} G$.

- H is an *(weak) immersion* if it can be obtained from a subgraph of G by lifting some edges[3]. We denote it by $H \leq_{\text{im}} G$.

Although these definitions of minors, contractions and immersions are very intuitive and, perhaps, easier to grasp, sometimes it is much more convenient to use the – more formal – definitions given in Section 6.2 of Chapter 6.

DEFINITION 2.3.1. We say that a graph G is *H-minor-free* (*H-topological-minor-free*) if it does not contain H as a minor (topological-minor). We also say that a graph class $\mathcal{C}$ is *H-minor-free* (*H-topological-minor-free*) if all of its members are H-minor-free (H-topological-minor-free).

[3]In other words: If there is an injective mapping $f : V(H) \to V(G)$ such that, for every edge $\{u, v\} \in E(H)$ there is a path from $f(u)$ to $f(v)$ in G, and for any two distinct edges of $E(H)$ the corresponding paths in G are *edge-disjoint*. If, in addition, these paths are *internally disjoint from* $f(V(H))$, then H is a *strong immersion of* G.

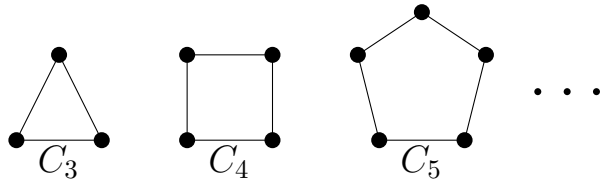


Figure 2.4: The set $\{C_i \mid i \geq 1\}$ of all cycles is an infinite anti-chain for $\leq, \leq_{\text{in}}$.

2.4 Well Quasi Orderings and Obstruction Sets

In the previous Section we defined six relations between graphs. These relations are *quasi-orderings*, in other words, *reflexive* and *transitive*. Some of them are called *well-quasi-orderings*, not according to someone's preferences, but according to – loosely speaking – the extent in which they are "partial". The following definition will make this make sense:

Definition 2.4.1. A quasi-ordering $\preceq$ is a *well-quasi-ordering* on a set $\mathcal{X}$ (of graphs in our case) if and only if for every infinite sequence $x_0, x_1, \ldots$ in $\mathcal{X}$ there exist two elements, say x_i and x_j, such that $x_i \preceq x_j$.

One can prove that a $\preceq$ is well-quasi-ordering on $\mathcal{X}$ if and only if $\mathcal{X}$ contains neither an *infinite anti-chain*, i.e., an infinite set of elements not comparable under $\preceq$, nor an infinite sequence $x_0, x_1, \ldots$ such that $x_0 \succ x_1 \succ \cdots$ (in other words an *infinite strictly decreasing sequence*). Using this proposition one can show that the class of all graphs, denoted by $\mathcal{G}$, is *not* well-quasi-ordered under $\leq, \leq_{\text{in}}$ (Figure 2.4), $\leq_{\text{c}}$ (Figure 2.5) and $\leq_{\text{tp}}$ (Figure 2.6).

Definition 2.4.2. Let $\mathcal{C} \subseteq \mathcal{G}$ be a class of graphs and $\preceq$ a quasi-ordering on graphs. $\mathcal{C}$ is $\preceq$*-closed* (or *closed under taking of* $\preceq$) if and only if for every $G \in \mathcal{C}$ and $H \in \mathcal{G}$, with $H \preceq G$, it also holds that $H \in \mathcal{C}$.

We will focus mainly on *minor-closed*, *contraction-closed* and *immersion-closed* graph classes. An example of a graph class closed under

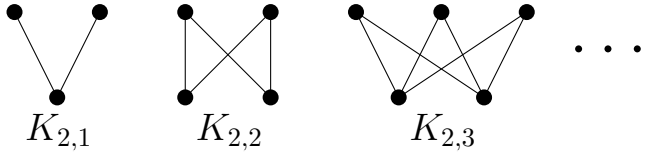


Figure 2.5: The set $\{K_{2,i} \mid i \geq 1\}$ is an infinite anti-chain for $\leq_{\mathrm{c}}$.

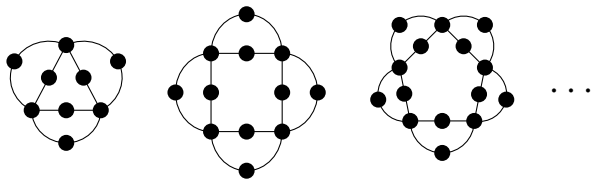


Figure 2.6: An antichain for $\leq_{\mathrm{tp}}$.

all these relations is the class of *planar graphs* that will be defined shortly (and redefined in Section 9.1.4 of Chapter 9).

Definition 2.4.3. Let $\mathcal{C}$ be a graph class closed under taking of $\preceq$. We denote by $\mathsf{obs}_{\preceq}(\mathcal{C})$ the set of all graphs in $\mathcal{G} \setminus \mathcal{C}$ that are minimal with respect to $\preceq$, and we call this set *obstruction set* (or just *obstructions*) for $\mathcal{C}$ under $\preceq$.

Some few words to make the name used for $\mathsf{obs}_{\preceq}(\mathcal{C})$ clear: Given a graph G, as $\mathcal{C}$ is $\preceq$-closed, if we can find a graph $O \in \mathsf{obs}_{\preceq}(\mathcal{C})$ such that $O \preceq G$, this will verify that $G \notin \mathcal{C}$. Therefore, we can check if a graph belongs to $\mathcal{C}$ using the graphs of $\mathsf{obs}_{\preceq}(\mathcal{C})$. This test by no means constitutes an algorithm, as $\mathsf{obs}_{\preceq}(\mathcal{C})$ may be infinite (Example 2.4.1), so we will have to make infinite many checks. Furthermore, even when this set is finite, there may not exist a polynomial time algorithm checking if two graphs are related under $\preceq$, thus, this test will not be efficient.

Let us see some examples of obstraction sets.

Example 2.4.1. Let $\mathcal{T}$ be the graph class of all trees. As we saw before, a graph is a tree if and only if it does not contain a cycle as a subgraph

(or induced subgraph). Therefore, the obstruction set of $\mathcal{T}$ under $\leq$ ($\leq_{\text{in}}$ respectively) is the set $\{C_i \mid i \geq 1\}$ of all cycles (see Figure 2.4), an *infinite set*.

Example 2.4.2. The obstruction set of $\mathcal{T}$ under $\leq_{\text{m}}$ contains only the graph C_3. First observe that every graph in $\mathcal{G} \setminus \mathcal{T}$ contains a cycle, say C, as a minor. We can further contract the edges of C until we obtain C_3. This is the minor minimal graph in $\mathcal{G} \setminus \mathcal{T}$ because if we delete a vertex or an edge, or contract an edge, we will obtain a tree (a path of length 2 in the first two cases and a path of length 1 in the last case).

2.5 Wagner's (?) Conjecture

All the observations discussed in the previous Section would not be of much use – in many respects – if it wasn't for *Graph Minors* ([65–87]), a much celebrated series of papers written by Neil Robertson and Paul Seymour. In this series, working towards proving *Wagner's Conjecture*, the authors introduced many useful tools that eventually, forever changed Graph Theory and Combinatorics.

We start this Section by stating Wagner's Conjecture:

Conjecture 2.5.1 ([4]Wagner, maybe in 1970 and perhaps in [93])**.** *The class $\mathcal{G}$ of all graphs is well-quasi-ordered under the minors relation.*

Notice that if $\mathcal{G}$ does not contain an infinite anti-chain, with respect of $\leq_{\text{m}}$, or an infinite strictly decreasing sequence, as this conjecture implies, then for every subclass $\mathcal{C} \subseteq \mathcal{G}$ closed under taking of minors, the class $\mathcal{G} \setminus \mathcal{C}$ would contain a finite number of minor-minimal graphs. Thus, the following is a direct corollary of this conjecture:

[4]Now known as the the *Robertson – Seymour Theorem* (see Theorem 2.6.1).

COROLLARY 2.5.1. For every minor-closed graph class $\mathcal{C}$ the obstruction set $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathcal{C})$ is finite.

Actually, Claus Wagner insisted that he never made this conjecture! Of course he was familiar with this problem and talk about it with his students back in the 60's [13]. The main reason why we attribute this conjecture to him is because Robertson and Seymour do so (in [84] they cited [93]).

The story starts in the 30's, with Wagner's PhD thesis, where he was trying to prove the *Four Color Theorem*:

> *Given a map (formally defined as a separation of the plane into regions) we can color all countries (the regions) so that every two adjacent countries have different colors, using at most four colors.*

To define this Theorem formally we need to introduce the notions of *planar graphs* and *graph coloring*.

DEFINITION 2.5.1. A *plane graph* is a pair $\Gamma = (V, E)$ of sets where:

1. $V \subseteq \mathbb{R}^2$,
2. every $e \in E$ is an arc between two points of V,
3. different arcs of E have different endpoints, and
4. the *interior* of an arc $e \in E$, i.e., the set consisted of every point belonging to e other than the two endpoints, does not contain a point in V or a point of an other arc of E.

Notice that every plane graph $\Gamma = (V, E)$ defines a (abstract) graph, denoted by $G(\Gamma)$, by ignoring the position of the points in V and identifying the arcs of E by the set of their endpoints. For simplicity, we will refer to the points of V as vertices and to the arcs of E as edges.

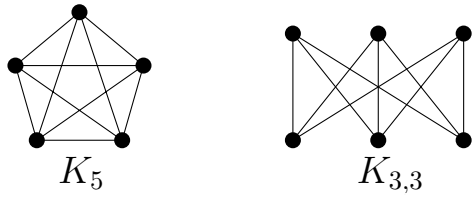


Figure 2.7: The two obstructions for planar graphs.

Definition 2.5.2. For every plane graph $\Gamma = (V, E)$ we call the regions of $\mathbb{R}^2 \setminus V \cup (\bigcup_{e \in E} e)$ *faces* of Γ and we denote this set by $F(\Gamma)$. The only unbounded face in $F(\Gamma)$ is the *outer face*, hence the other faces are *inner faces*.

Definition 2.5.3. An (abstract) graph G is *planar* if and only if there exists a plane graph Γ such that $G(\Gamma)$ is isomorphic to G.

Definition 2.5.4. A *(proper) vertex coloring* (or just *coloring*) of a graph G that uses k *colors* is a function $f : V(G) \to [k]$ such that for every edge $\{u, v\} \in E(G)$, $f(u) \neq f(v)$.

The formal definition of the Four Color Theorem is the following:

Theorem 2.5.1 (Four Color Theorem[5])**.** *Every planar graph is 4-colorable.*

Wagner's approach was to try to completely bypass topology and give a combinatorial proof. He tried to classify the K_5-minor free graphs and prove that they are 4-colorable. As the class of K_5-minor-free graphs contains all planar graphs (a consequence of Theorem 2.5.2), this would imply that all planar graphs are 4-colorable. Wagner's efforts accumulated to the following theorem that was proven in 1937 [94] and is – rightfully – bearing his name.

[5]First proven in 1976 by Kenneth Appel and Wolfgang Haken with the use of a computer [25–29].

Theorem 2.5.2 (Wagner, 1937 [94]). *A graph G is planar if and only if neither $K_5 \leq_{\mathrm{m}} G$ nor $K_{3,3} \leq_{\mathrm{m}} G$ (Figure 2.7).*

Unfortunately for Wagner, this theorem succeed in characterizing K_5-minor-free graphs but failed to take "planarity", and Topology in general, out of the picture. Wagner's "failure" had great consequences in the development of Graph Theory. First of all, it inspired *Hardwiger's Conjecture*, still "one of the deepest unsolved problems in graph theory" [38]:

Conjecture 2.5.2 (Hardwiger, 1943 [45]). *If all colorings of a graph G use at least k colors, then $K_k \leq_{\mathrm{m}} G$.*

It also inspired the notion of *Tree-decompositions* (Definition 4.1.3) a central notion in modern Graph Theory, and fundamental for the proof of *Robertson–Seymour Theorem* (Theorem 2.6.1).

Last but not least, consider the class of all planar graphs, say $\mathcal{P}$. It is straightforward to see that $\mathcal{P}$ is closed under minors. What Wagner proved was that the obstruction set $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathcal{P})$ is finite and contains only two graphs.

Notice that this is an extremely elegant way to define planar graphs: not only it is simple but also it is purely combinatoric. This definition of a graph class through the set of its "forbidden graphs", known as *Forbidden graph characterization* or *Kuratowski characterization* after *Kazimierz Kuratowski*[6], can be considered as the first results in the field of Graph Minors. It motivated researchers to find such characterizations for other graph classes, using the quasi-orderings defined previously. We will talk more about this in the following Section.

[6]Kuratowski proved in 1930 a lighter version of Theorem 2.5.2 for topological minors [51]. But, since this was the first forbidden graph characterization, such characterization often bears Kuratowski's name. Interestingly, the same result had been already proven independently by the Soviet Mathematician *Lev Semyonovich Pontryagin* (Лев Семёнович Понтрягин) around 1927. However, the proof of Pontryagin has never been published. This recommends that the – somewhat more politically correct – way to name this theorem is "Kuratowski-Pontryagin Theorem".

2.6 Graph Minors

Robertson and Seymour published the proof of Wagner's Conjecture in 2004. It is known as *Robertson–Seymour Theorem* or *Graph Minors Theorem*.

THEOREM 2.6.1 (*Robertson–Seymour Theorem*, 2004 [84])**.** *The class $\mathcal{G}$ of all graphs is well-quasi-ordered under the minor relation.*

This theorem immediately guarantees that there exists a *finite* forbidden minors characterization for every graph class, closed under taking of minors. But, given a minor closed graph class $\mathcal{C}$ is it possible to find this characterization? In other words, assume that we are given a finite representation of $\mathcal{C}$, can we compute $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathcal{C})$? In this Section we will discuss some of the algorithmic implications of this theorem and try to give an answer to this question. This will pave the way for the following chapter on *Parameterized Complexity*.

Before getting any further, we have to stress that we assume the reader is familiar with the notion of *algorithm*, the *time complexity* of algorithms and the *big O notation*.

In this thesis we will heavily use "tools" from the Graph Minors "toolset", more specifically, we will use some structures and parameters defined in the Graph Minors series, alongside with some of their major results. The first algorithmic result we will need was published in 1995 [77]. Consider the following two problems:

H-MINOR-CONTAINMENT	
Input:	A graph G.
Question:	Is $H \leq_{\mathrm{m}} G$?

$\mathcal{C}$-MEMBERSHIP	
Input:	A graph G.
Question:	Is $G \in \mathcal{C}$?

THEOREM 2.6.2 (Robertson and Seymour, 1995 [77]). *There exist an algorithm – with running time $O(n^3)$ – deciding the H-*MINOR-CONTAINMENT*, where $n = n(G)$.*[7]

Combining Theorem 2.6.1 with Theorem 2.6.2 one can see that, if $\mathcal{C}$ is a minor-closed graph class, the $\mathcal{C}$-MEMBERSHIP problem has an $O(n^3)$ algorithm, provided that $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathcal{C})$ is known (just run $|\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathcal{C})|$-times the algorithm for H-MINOR-CONTAINMENT problem, one for each $H \in \mathsf{obs}_{\leq_{\mathrm{m}}}(\mathcal{C})$, and, if all answers are *negative* then $G \in \mathcal{C}$, otherwise $G \notin \mathcal{C}$). The only thing missing to have an affirmative answer to the question posed in the beginning of this Section is an algorithm that will produce the obstruction set. Here is where things stop being so favourable for us: Theorem 2.6.1 is not – and cannot be – constructive [88]. To formally explain this, we have to dive deep into Logic and this – almost certainly – will take us off our track. Therefore, we will only say that, as the *Axiom of Choice* is used in the proof of the Graph Minors Theorem[8] we cannot draw an algorithm from it that, taking a minor closed graph class $\mathcal{C}$ as input, will compute $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathcal{C})$. In [88] Friedman, Robertson and Seymour showed that we cannot prove the Graph Minors Theorem without using certain variants of the axiom of choice, hence we will never find such an algorithm in general.

Although we do not know a – universally applicable – method to compute the obstruction sets, our research is by no means hopeless, as for some specific graph classes this may be possible by applying ad hoc methods.

In Section 2.4 we saw that a "Graph Minors" type of theorem is not possible for $\leq, \leq_{\mathrm{in}}$ and $\leq_{\mathrm{c}}$, as $\mathcal{G}$ is not well-quasi-ordered under these relations. But what about the immersion relation?

CONJECTURE 2.6.1 (Weak Immersions Conjecture, Nash-Williams, 1963 [91]). *The class $\mathcal{G}$ of all graphs is well-quasi-ordered under the weak im-*

[7] A reminder: We will occasionally use n instead of $n(G)$ and m instead of $m(G)$.

[8] Actually *Kruskal's Tree Theorem* is used, which in turn uses the Axiom of Choice.

mersion relation.

Conjecture 2.6.2 (Strong Immersions Conjecture, Nash-Williams, 1965 [90])**.** *The class $\mathcal{G}$ of all graphs is well-quasi-ordered under the strong immersion relation.*

Both these conjecture was made by Crispin St. John Alvah Nash-Williams in the sixties. The first one was resolved after 45 years in the last paper of the Graph Minors series [87]. It is indeed true. For the second one, in [87] Robertson and Seymour remark that:

> *"It seemed to us at one time that we had a proof of the stronger [conjecture], but even if it was correct it was very much more complicated, and it is unlikely that we will write it down."*

Therefore, we reached the same conclusion as in the minor relation:

Corollary 2.6.1. For every immersion-closed graph class $\mathcal{C}$ the obstruction set $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C})$ is finite.

One year after the publication of the proof of this conjecture, a cubic time algorithm was given by Grohe, Kawarabayashi, Marx, and Wollan for the following problem:

H-Immersion-Containment	
Input:	A graph G.
Question:	Is $H \leq_{\text{im}} G$?

Theorem 2.6.3 (Grohe, Kawarabayashi, Marx, and Wollan, 2011 [179])**.** *There exist an algorithm – with running time $O(\|G\|^3)$ – deciding the H-*Immersion-Containment.

This theorem applies only for the weak immersions relation[9] and is an almost direct corollary of the following.

[9]The authors of [179] conjecture that there exists such an algorithm for the strong immersion relation as well. Yet, it still eludes us.

THEOREM 2.6.4 (Grohe, Kawarabayashi, Marx, and Wollan, 2011 [179])**.** *For every fixed graph H, there exist an algorithm – with running time $O(|G|^3)$ – that decides if H is a topological-minor of G.*

Thus, for an immersion-closed[10] graph class $\mathcal{C}$ there exist a $O(n^3)$ time algorithm deciding the $\mathcal{C}$-MEMBERSHIP problem, under the condition that $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C})$ is known.

Much effort has been made to find the necessary conditions to make the computation of the forbidden minors, or forbidden immersions, possible, for a variety of graph classes (a good start for those who want to learn more about this are [52, 123, 125, 127, 192, 193]). In Chapter 6 we will argue that when $\mathcal{C}$ is immersion-closed, *MSO expressible* and an explicit bound on the *treewidth* of the $\leq$-minimal graphs of $\mathcal{G} \setminus \mathcal{C}$ can be computed, there *exists* an algorithm computing $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C})$. There are still some things missing before we are able to formally express this result. In the following couple of Chapters we plan to complete the definitions required to do it.

2.7 Logic

Let us recall some definitions from Monadic Second-Order Logic (MSO). An extended introduction to Logic can be found in [186, 188][11]. We will take a shortcut and immediately present the syntax of MSO of graphs.

2.7.1 Monadic Second-Order logic

DEFINITION 2.7.1. We call *signature* $\tau = \{R_1, \ldots, R_n\}$ any finite set of relation symbols R_i of any (finite) arity denoted by $\text{ar}(R_i)$.

[10] For now on, when we just say immersion we mean weak immersion.

[11] It is highly advised for the reader to revise the basic notions of logic before getting any further in this Section.

DEFINITION 2.7.2. A *τ-structure* $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, \ldots, R_n^{\mathfrak{A}})$ consists of a finite universe A, and the interpretation of the relation symbols R_i of τ in A, that is, for every i, $R_i^{\mathfrak{A}}$ is a subset of $A^{\text{ar}(R_i)}$.

The syntax of Monadic Second-Order logic of graphs includes the logical connectives $\vee$, $\wedge$, $\neg$, $\leftrightarrow$, $\rightarrow$, variables for vertices, edges, sets of vertices, and sets of edges, the quantifiers $\forall$, $\exists$ that can be applied to these variables, the signature $\tau_{\mathcal{G}} = \{V, E, I\}$ where:

1. V has arity one and represents the set of vertices of a graph G,
2. E has arity one and represents the set of edges of G, and
3. $I = \{(v, e) \mid v \in e \text{ and } e \in E(G)\}$ the incidence relation,

and equality of variables. We denote the language of graphs by $L^{\mathcal{G}}$.

DEFINITION 2.7.3. An *MSO-formula* is defined recursively from *atomic formulas* (i.e., expressions of the form $R_i(x_1, x_2, \ldots, x_{\text{ar}(R_i)})$ or of the form $x = y$, where x_j, $j \leq \text{ar}(R_i)$, x and y are variables) by using the *Boolean connectives* $\neg, \wedge, \vee, \rightarrow$, and existential ($\exists$) or universal ($\forall$) quantification over individual variables and sets of variables.

Notice that in $L^{\mathcal{G}}$ the atomic formulas are of the form $V(u), E(e)$, $X(u)$, $Y(e)$, and $I(u, e)$, where u and e are vertex and edge variables respectively, X is a *vertex-set variable*, and Y is an *edge-set variable*. Furthermore, quantification takes place over vertex or edge variables or vertex-set or edge-set variables.

DEFINITION 2.7.4. A *graph structure* $\mathfrak{G} = (V(G) \cup E(G), V^{\mathfrak{G}}, E^{\mathfrak{G}}, I^{\mathfrak{G}})$ is a $\tau_{\mathcal{G}}$-structure, which represents the graph $G = (V, E)$.

From now on, we will abuse the notation and treat G and $\mathfrak{G}$ as if they were the same structure.

Definition 2.7.5. A graph class $\mathcal{C}$ is *MSO-definable* if there exists an MSO-formula $\phi_{\mathcal{C}}$, in the language of graphs, such that

$$G \in \mathcal{C} \text{ if and only if } G \models \phi_{\mathcal{C}},$$

i.e., $\phi_{\mathcal{C}}$ is true in the graph structure $\mathfrak{G}$ representing G (if this is the case we will say that G is a *model* of $\phi_{\mathcal{C}}$).

To make this clear let us see an example (which we will later use in the proofs of Lemma 6.2.2 and Observation 6.3.1):

Lemma 2.7.1. The class of graphs that contain a fixed graph H as an immersion is MSO-definable.

Proof. Let $V(H) = \{v_1, v_2, \ldots, v_n\}$ and $E(H) = \{e_1, e_2, \ldots, e_m\}$. Let also ϕ_H be the following formula.

$$\begin{aligned}\phi_H \;=\; & \exists E_1, E_2, \ldots, E_m \, \exists x_1, x_2, \ldots, x_n \Big[\Big(\bigwedge_{i \in [n]} V(x_i) \Big) \\ & \wedge \Big(\bigwedge_{j \in [m]} E_i \subseteq E(H) \Big) \\ & \wedge \Big(\bigwedge_{i \neq j} x_i \neq x_j \Big) \wedge \Big(\bigwedge_{p \neq q} E_p \cap E_q = \emptyset \Big) \\ & \wedge \Big(\bigwedge_{e_r = \{v_k, v_l\} \in E(H)} \text{path}(x_k, x_l, E_r) \Big) \Big],\end{aligned}$$

where $\text{path}(x, y, Z)$ is the MSO-formula stating that:

"The edges in Z form a path from x to y."

This can be done by saying that every vertex v incident to an edge in Z is either incident to exactly two edges of Z or to exactly one edge, with the further condition that $v = x$ or $v = z$ (notice that these conditions also

imply the connectivity of the graph induced by the edges of Z). Thus, path(x, y, Z) can be expressed in MSO by the following formula:

$$\begin{aligned}
\text{path}(x, y, Z) \quad = \quad & [(x \neq y) \wedge \exists p, q(Z(p) \wedge Z(q) \wedge I(x, p) \wedge I(y, q) \\
& \wedge \forall p' \in Z(I(x, p') \rightarrow p = p') \\
& \wedge \forall q' \in Z(I(y, q') \rightarrow q = q')) \\
& \wedge \forall w(V(w) \wedge w \neq x \wedge w \neq y \\
& \wedge \exists q_1(Z(q_1) \wedge I(w, q_1)) \\
& \rightarrow \exists q_2, q_3(Z(q_2) \wedge Z(q_3) \wedge q_2 \neq q_3 \\
& \wedge I(w, q_2) \wedge I(w, q_3))) \\
& \wedge \forall p_1, p_2, p_3(Z(p_1) \wedge Z(p_2) \wedge Z(p_3) \\
& \wedge \exists m(V(m) \wedge I(u, p_1) \wedge I(u, p_2) \wedge I(u, p_3)) \\
& \rightarrow \bigvee_{i \neq j} (p_i = p_j))]
\end{aligned}$$

It is easy to verify that ϕ_H is the desired formula. □

2.7.2 Counting Monadic Second-Order logic

If, in addition to the features of Monadic Second-Order logic, we also have atomic formula "testing" whether the cardinality of a set is equal to q modulo r, where q and r are integers such that $0 \leq q < r$ and $r \geq 2$, then we have an extension of MSO, called *Counting Monadic Second-Order logic (CMSO)*.

More precisely, CMSO is MSO enriched with the following atomic formula for every set S:

$$\mathsf{card}_{q,r}(S) = \textbf{true} \text{ if and only if } |S| = q \bmod r$$

As far as "expressibility" is concerned, CMSO is more powerful than MSO, as there exist graph properties that can be defined in CMSO but not in MSO [191].

Definition 2.7.6. A graph class $\mathcal{C}$ is *CMSO-definable* if there exists a CMSO-formula $\phi_{\mathcal{C}}$, in the language of graphs, such that $G \in \mathcal{C}$ if and only if $G \models \phi_{\mathcal{C}}$.

We will see some examples of CMSO formulas in the following Section (Example 3.4.1 and 3.4.2). More information about CMSO (and MSO in general) can be found in [31, 190, 191] and of course [184].

CHAPTER 3

PARAMETERIZED COMPLEXITY

The meta-algorithmic result of the existence of a cubic time algorithm deciding the $\mathcal{C}$-MEMBERSHIP problem for a minor-closed graph class $\mathcal{C}$, broadened the perspectives towards the understanding of NP-hard problems. It was actually at that point when it became clear that, with the introduction of more variables when measuring the complexity of a problem, there seems to be "*different levels of hardness*" between problems in NP [138]. To set a working hypothesis let us define two more problems.

A *vertex cover* of a graph G is a set $S \subseteq V(G)$ such that for very edge $\{u, v\} \in E(G)$ either $u \in S$ or $v \in S$.

k-VERTEX COVER	
Input:	A graph G.
Question:	Does G contain a vertex cover of size k?

Recall the definition of coloring (Definition 2.5.4).

k-Coloring	
Input:	A graph G.
Question:	Can G be colored using k colors?

Notice that, for the k-Vertex Cover problem, the class of graphs admitting a vertex cover of size at most k is closed under taking of minors. Therefore, for every fixed k there is a cubic time algorithm deciding whether a graph has a vertex cover of size k (of course there exist way better algorithms for this problem, the algorithms in [161, 162] for instance). However, no similar result can be expected for the k-Coloring problem, as it is known to be NP-hard for every fixed $k \geq 3$ [187]. Hence, although both problems belong to NP, the k-Coloring problem seems to be inherently more difficult than the k-Vertex Cover problem.

The observation of this gap in the time complexity of NP-hard problems facilitated the development of *Parameterized Complexity Theory* [139–143] by Fellows and Downey [167, 168] , which has proven to be a very powerful theory and has majorly advanced during the past decades (see [24, 153, 154, 164, 165]).

Our starting point for this Chapter will be *Classic Complexity Theory*.

3.1 Classic Complexity Theory

Let Σ^* be the set of all *strings* or *words* of a finite *alphabet* Σ. A *classic problem* Π is a *language* over Σ, that is a set $\Pi \subseteq \Sigma^*$. The *inputs* or *instances* of Π are the words of Σ^*. We call an input $w \in \Sigma^*$ Yes-*instance* if $w \in \Pi$ and No-*instance* if $w \notin \Pi$.

As we are mainly interested in *graph problems* we will implicitly use a representation of graphs and a function mapping this representation to words of an alphabet. For instance, we can use a *binary representation* of a graph. In this way we can think of graphs as being words in $\{0, 1\}^*$ and

problems as being subsets of $\mathcal{G}$.

To make the notation more intuitive, we will describe classic problems less formally, following the way we have defined problems so far:

Π	
Input:	$w \in \Sigma^*$.
Question:	Is $w \in \Pi$?

Whenever we say that there exists an algorithm solving a problem Π, we mean that there exist a *Turing machine* that takes as input the input of Π and returns a "Yes" or "No" answer to the question of the problem.

The *time complexity* of an algorithm is the number of steps needed for the corresponding Turing machine to finish its computation and output an answer. To establish a measure of time complexity we count this number of steps as a function of the size of the input word, i.e., the number of its symbols. For a word w we will denote its size as $|w|$. In classic Complexity Theory we are interested in the *worst-case analysis* of the time complexity of an algorithm. Therefore, we only focus on the time required for an algorithm to give us an answer in the worst possible input.

To express how well an algorithm performs, as far as time complexity is concerned, we will use the *Big O* notation.

As already mentioned, in this thesis we examine only graph problems. Therefore, it is more suited to measure the performance of an algorithm as a function of the size of the graph that was given to us as input. If not otherwise stated, the input size will be the number of vertices of the input graph.[1]

The biggest open question in classical Complexity Theory is whether

[1]Of course one cannot ignore the fact that a graph may contain n vertices but in reality needs approximately n^2 symbols to be expressed in some data structure. Moreover, each different data structure has its own advantages and disadvantages when it comes to manipulating a graph. We will avoid getting that deep in data structures as we are only interested in how algorithms perform in a higher level.

P is equal to NP or not. P contains the problems that can be solved in *polynomial time*, i.e., needing $O(n^c)$ time steps in order to be solved, for some constant c. NP on the other hand, contains the problems that can be "verified" in polynomial time, that is, if someone were to give us a certificate of a solution we could then run a polynomial time algorithm and compute the answer of the problem. Up to now, no-one knows the answer, although the vast majority[2] of computer scientists is convinced that these sets are not equal and that there exist problems in NP that do not admit polynomial time algorithms (obviously $\mathsf{P} \subseteq \mathsf{NP}$).

There exist some problems in NP that are – in a sense – the most difficult of all: If one of these problems were to admit a polynomial time algorithm, then all problems in NP would also admit such an algorithm. These problems are called NP-hard. The NP-hard problems that belong to NP are called NP-complete problems.

Polynomial time algorithms are commonly fast and very useful for the applications. Hence, when a problem is proven to be NP-complete this means that its almost certainly hard – at least all the great minds of Theoretical Computer Science believe so – and no-one can find a feasible solution to it (see pages 2,3 of [187]). Researchers came to terms with this fact and try hard to find other methods to tackle these problems. A paradigm of a way to look the difficulty of an NP-complete problem in the eyes and try to find algorithms that will perform good despite not being polynomial, will be described in the following Section.

3.2 Fixed Parameter Tractable Algorithms

The framework of classical complexity theory has a drawback. In many problems, when measuring their difficulty only in terms of the input size,

[2]We stress here that the "vast majority vote" does not imply "truth" in mathematics (while this might be the case in other circumstances).

we may overlook some structural properties of the input. Using these structural properties we could be able to design feasible algorithms. This becomes clearer if we consider graph problems, and the fact that the "size" of a graph tells almost nothing about the properties of the graph. To back this thought up, we are going to define a different way of measuring the complexity of a problem. First, we will distinguish a secondary measurement of the input that governs the computation complexity of the problem. Then, we will define the time complexity of an algorithm in a multivariate way, meaning that we will count the time steps needed for an algorithm to output a result as a function, not only of the size of the input, but also as a function of these measurements. The way these two variables are correlated in the time complexity of an algorithm yields different complexity classes.

Let us introduce some notation.

Definition 3.2.1. Let Σ be a finite alphabet. A polynomial time computable function[3] $\kappa : \Sigma^* \to \mathbb{N}$ is a *parameterization* of Σ^*.

In this new theory we consider problems where we are given as input a pair consisted of an input in the classic sense, a word, and the value of the parameterization, an integer [4], and we are asked to decide whether some property holds. These type of problems can be formally defined as follows.

Definition 3.2.2. Let Σ be a finite alphabet. A *parameterized problem* over Σ consists of a classic problem $\Pi \subseteq \Sigma^*$ and a parameterization κ of Σ^*. We will denote this problem as p-Π to indicate that this is the parameterized version of Π. The instances of p-Π are pairs of the form $(w, \kappa(w)) \in \Sigma^* \times \mathbb{N}$.

[3] By that we mean that there exists a (known) Turing Machine that can compute the output of this function in polynomially-many steps (on the size of the input word).

[4] We will refer to this parametrization as the *parameter* of the problem.

Let p-Π be a parameterized problem. The Yes-*instance* of p-Π are the pairs $(w, \kappa(w)) \in \Sigma^* \times \mathbb{N}$ that belong to p-Π and the No-*instance* all other pairs, as usual. In what follows, if there is no particular reason to mention the function κ, we will denote an instance as (w, k), meaning that $k = \kappa(w)$.

We will measure the time complexity of a parameterized problem as a function of the input size and the value of the parameter. Similarly to the case of Classic Complexity, we are interested only in the worst-case analysis of algorithms. To make the two parts of a parameterized problem clear, we will use the following representation:

p-Π	
Input:	$(w, \kappa(w)) \in \Sigma^* \times \mathbb{N}$.
Parameter:	$\kappa : \Sigma^* \to \mathbb{N}$.
Question:	Is $(w, \kappa(w)) \in p$-Π?

The idea behind measuring the complexity of a problem as a function of the input size and the parameter, stems from the fact that in many problems where the parameter is small – a lot smaller than the size of the input – simple exponential algorithms (in the classic complexity set up) can perform exceptionally well. For instance, let us parametrize the k-Vertex Cover problem by the constant function k. The best known algorithm solving this problem runs in $O(1.2738^k + k \cdot n)$ steps [161, 162]. For small values of k this is an extremely feasible algorithm.

If you are not still persuaded – and you mustn't – think of the possible ways the parameter and the input size can be combined when measuring the complexity of a problem. For running times such as $O(f(k) \cdot n^c)$ or even $O(n^{f(k)})$, where f is just a computable function and c a constant, for every single value of k, the parameter, we have a polynomial algorithm on n, the input size. The real "tough" problems in parametrized complexity have running times such as $O(k^n)$, where even for constant k we get an

exponential algorithm.

I think we had enough hand having so far (too much for my likings), so let us go through the definitions.

Definition 3.2.3. Let p-Π be a parametrized problem over Σ and $\kappa : \Sigma^* \to \mathbb{N}$ its parameterization. An algorithm $\mathcal{A}$ is *fixed parameter tractable*, or an FPT-*algorithm*, if there exist a computable function $f : \mathbb{N} \to \mathbb{N}$ and a polynomial p such that for every $w \in \Sigma^*$ the running time of $\mathcal{A}$ on input w is:

$$O(f(\kappa(w)) \cdot p(|w|))$$

That is potentially exponential on the parameter but polynomial on the size of the input.

Definition 3.2.4. A parametrized problem p-Π is *fixed parameter tractable* if it admits an FPT-algorithm deciding it.

For our convenience, we call such problems FPT-*problems*. To add more to the overuse of FPT:

Definition 3.2.5. We define FPT to be the class of all FPT-problems.

To get a grasp on these definitions we need some examples. Let us review Paragraph 2.6 and give parametrizations of the problems described there. First of all, we have to stress that

Minor-Containment	
Input:	Two graphs H and G.
Question:	Is $H \leq_{\text{m}} G$?

and

Immersion-Containment	
Input:	Two graphs H and G.
Question:	Is $H \leq_{\text{im}} G$?

are NP-complete problems[5]. We will give a parameterization of these problems by defining $\kappa : \mathcal{G} \to \mathbb{N}$ to be the number of vertices of H:[6]

p-MINOR-CONTAINMENT	
Input:	Two graphs H and G.
Parameter:	$n(H)$.
Question:	Is $H \leq_{\text{m}} G$?

p-IMMERSION-CONTAINMENT	
Input:	Two graphs H and G.
Parameter:	$n(H)$.
Question:	Is $H \leq_{\text{im}} G$?

We saw that if $\mathcal{C}$ is a minor-closed (immersion-closed) graph class then $\mathsf{obs}_{\leq_{\text{m}}}(\mathcal{C})$ ($\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C})$ respectively) is a finite set of graphs, a set that characterizes $\mathcal{C}$. Let us focus first in the minor case and assume that the set $\mathsf{obs}_{\leq_{\text{m}}}(\mathcal{C}) = \{O_1, \ldots, O_{|\mathsf{obs}_{\leq_{\text{m}}}(\mathcal{C})|}\}$ is known. Moreover let us look deeper in the algorithm of Theorem 2.6.2:

THEOREM 3.2.1 (Robertson and Seymour, 1995 [77])**.** *There exist an $O(f(k) \cdot n^3)$ algorithm deciding the p-MINOR-CONTAINMENT problem, where $k = n(H)$.*

The function f in the running time of this algorithm is a horrible, yet computable function. Therefore, this algorithm is not feasible, but, as we saw before, it yields a cubic algorithm for the $\mathcal{C}$-MEMBERSHIP problem

[5]This can be proven in the following way: Take for instance H to be a cycle with $n(G)$ vertices. Then $H \leq_{\text{m}} G$ if and only if G contains a *Hamiltonian cycle*, thus the HAMILTONIAN CYCLE problem can be reduced to the MINOR-CONTAINMENT problem (for more informations check [187]). To prove that MINOR-CONTAINMENT $\in$ NP take as certificate the function ψ of Definition 6.2.2.

[6]Also we have two graphs as input, in essence two words. To be consistent with our framework, we have to code these two words as one. Then we decode it to take the first word in order to compute the parameter.

with $O(|\mathsf{obs}_{\leq_{\mathtt{m}}}(\mathcal{C})| \cdot f(k) \cdot n^3)$ running time, where $k = \max\{n(O) \mid O \in \mathsf{obs}_{\leq_{\mathtt{m}}}(\mathcal{C})\}$.

The same thing holds for the immersion case:

Theorem 3.2.2 (Grohe, Kawarabayashi, Marx, and Wollan, 2011 [179])**.** *There exist an $O(f(k)\cdot n^3)$ algorithm deciding the p-*Immersion-Containment *problem.*

Again, this algorithm was not meant to be used in real life applications, as f is rather cumbersome, but it is of great significance from a theoretical point of view.

For the purposes of this thesis we will only concider *parameterized graph problems*:

Definition 3.2.6. A parameterized problem p-Π is a *(parameterized) Graph problem* if in every instance $(w, k) \in \Sigma^* \times \mathbb{N}$, w encodes a graph. For simplicity we will denote the instances of such problems by $(G, k) \in \mathcal{G} \times \mathbb{N}$, where G is the graph encoded by the word of the input.

Every so often we have to focus on problems defined on graphs that have specific properties (planar graphs for instance). Therefore, we must define the restriction of a problem to a ceratin graph class.

Definition 3.2.7. Let $\mathcal{C} \subseteq \mathcal{G}$ be a class of graphs. The *restriction* of a parameterized problem p-Π to $\mathcal{C}$, is defined as

$$p\text{-}\Pi \Cap \mathcal{C} = \{(G, k) \in \mathcal{G} \times \mathbb{N} \mid (G, k) \in p\text{-}\Pi \text{ and } G \in \mathcal{C}\}.$$

3.3 Kernelization

In this Section we will define the class FPT again, this time using *kernels*. To give you an idea of what we are about to present, *Kernelization* is a type of pre-processing in the input, after which we have a reduced "equivalent"

instance, the kernel. This kernel, as the name suggests, is the hard part of the input, i.e., the part that contains all the information needed to decide if this instance is a Yes-instance or a No-instance. The fact that the size of the kernel is reduced (usually it is bounded by a polynomial on the value of the parameter) gives us the ability to solve the problem in FPT time.

Definition 3.3.1. Let p-Π be a parameterized problem over Σ and $\kappa : \Sigma^* \to \mathbb{N}$ its parameterization. Let also g be a computable, increasing function. We say that Π *admits a kernel of size* g if there exists a computable function (an algorithm if you prefer) $\mathcal{K} : \Sigma^* \to \Sigma^*$, called *kernelization (algorithm) for* p-Π, or, in short, a *kernelization*, that given $w \in \Sigma^*$ outputs, in time polynomial in $|w| + \kappa(w)$, a $w' \in \Sigma^*$ such that

(a) $w \in \Pi$ if and only if $w' \in \Pi$, and

(b) $\max\{|w'|, \kappa(w')\} \leq g(\kappa(w))$.

When $g(k) = k^{O(1)}$ or $g(k) = O(k)$ then we say that p-Π *admits a polynomial* or *linear kernel* respectively.

We often abuse the notation and call the output of a kernelization algorithm, the "reduced" equivalent instance $\mathcal{K}(w)$, also a kernel.

The following Theorem shows that a decidable problem[7] p-Π is FPT if and only p-Π admits a kernelization.

Theorem 3.3.1 (Theorem 1.39 of [142])**.** *For every parameterized problem* p-Π*, the following are equivalent:*

1. p-$\Pi \in$ FPT.
2. p-Π *is decidable and admits a kernelization.*

[7] Π is decidable if there exist a Turing machine that for every input $w \in \Sigma^*$ stops and returns Yes if $w \in \Pi$ and No if $w \notin \Pi$

Kernelization has evolved to a prominent area of parameterized computation and initiated numerous techniques and methodologies, especially for problems related to graphs [145, 147].

In Chapter 7 we will present a (meta-)theorem that proves the existence of linear kernels for a large class of problems. In addition to being linear, these kernels have the two properties defined in Definition 3.3.2 and 3.3.3. These properties are important for our purposes as they make the computation of obstruction sets possible, for graph classes defined form problems admitting such kernels. This is one of the extremely rare instances where kernels yield obstruction sets (the only other examples are [169, 171]).

Definition 3.3.2. Let p-Π be a parameterized graph problem and $\preceq$ a partial ordering relation on graphs. We say that a kernelization algorithm for p-Π is $\preceq$-*monotone* if, given an instance (G, k) it outputs an instance (G', k'), where $G' \preceq G$.

When taking as $\preceq$ the minor relation (as we plan to do in Chapter 7), we define *minor-monotone* kernelization algorithms.

Definition 3.3.3. Let p-Π be a parameterized problem. We say that a kernelization algorithm for p-Π is *parameter-invariant* if, given an instance (w, k) it outputs an instance (w', k'), where $k = k'$.

Being consistent with the aforementioned abuse of notation, we call the corresponding kernels $\preceq$-*monotone* and *parameter-invariant* respectively.

3.4 Optimization Graph Problems and Their Properties

The vast majority of problems we will consider in this thesis are *Optimization problems*, where we want to minimize or maximize a quantity. To formally define them we will need some definitions.

Definition 3.4.1. An *annotated graph* is a pair (G, S) where G is a graph and $S \subseteq V(G)$.

Definition 3.4.2. A *vertex subset certifying function* f (or certifying function for short) is a computable function which takes as input an annotated graph (G, S) and outputs true or false. We say that the function f is *CMSO-definable* if there is a CMSO-formula on annotated graphs ϕ such that $f(G, S) =$ **true** if and only if $(G, S) \models \phi$.

We can define in the same way *edge subset certifying functions* but we will not. We will focus on *vertex subset minimization or maximization problems*.

Definition 3.4.3. A *(vertex subset) minimization/maximization problem* p-Π is a parameterized problem on graphs for which there exists a certifying function f such that for every $(G, k) \in \mathcal{G} \times \mathbb{N}$ it holds that

$$(G, k) \in p\text{-}\Pi \iff \exists S \subseteq V(G) : |S| \lesseqgtr k \text{ and } f(G, S) = \textbf{true}$$

where $\lesseqgtr$ is interpreted as $\leq$ for minimization problems and as $\geq$ for maximization problems. Such problems are also called *optimization graph problems* (or just *optimization problems*, as we will only consider graph problems).

When the certifying function f is CMSO-definable, we say that p-Π is a MIN-CMSO problem or a MAX-CMSO problem, depending on whether p-Π is a minimization or a maximization problem. For simplicity, we will call such problems *CMSO-definable optimization problems*.

Example 3.4.1. The aforementioned – but not formally defined – parameterized version of k-Vertex Cover, where the parameter is k:

p-VERTEX COVER	
Input:	A graph G and a $k \in N$.
Parameter:	k.
Question:	Does G contain a vertex cover of size $\leq k$?

is a MIN-CMSO problem

This holds because p-VERTEX COVER is a minimization problem, and the function f that, given an instance (G, k), certifies whether a set $S \subseteq V(G)$ is a vertex cover of G of size at most k, can be defined in CMSO as follows:

$$\begin{aligned} f(G,S) &= [\forall e \forall u \forall v (E(e) \wedge I(u,e) \wedge I(v,e) \\ &\quad \rightarrow S(u) \vee S(v))] \\ &\quad \wedge \phi_{\leq k}(S) \end{aligned}$$

where

$$\phi_{\leq k}(S) = \mathbf{card}_{0,1}(S) \vee \mathbf{card}_{0,2}(S) \vee \cdots \vee \mathbf{card}_{0,k}(S),$$

i.e., $\phi_{\leq k}(S) = \mathbf{true}$ if and only if $|S| \leq k$.

EXAMPLE 3.4.2. An *independent set* of a graph G is a subset $S \subseteq V(G)$ where no two vertices $u, v \in S$ are connected with an edge. Let us define the following parameterized problem:

p-INDEPENDENT SET	
Input:	A graph G and a $k \in N$.
Parameter:	k.
Question:	Does G contain an independent set of size $\geq k$?

p-INDEPENDENT SET is a MAX-CMSO problem, as in this problem we want

to maximize the size of an independent set, and the function f which, given an instance (G, k), certifies whether a set $S \subseteq V(G)$ is an independent set of G of size at least k, can be defined in CMSO as follows:

$$\begin{aligned} f(G,S) = [&\forall u \forall v (V(u) \wedge V(v) \wedge S(u) \wedge S(v) \\ &\rightarrow \neg \exists e (E(e) \wedge I(u,e) \wedge I(v,e)))] \\ &\wedge \phi_{\geq k}(S) \end{aligned}$$

where

$$\phi_{\geq k}(S) = \neg \phi_{\leq k}(S) \vee \mathbf{card}_{0,k}(S),$$

i.e., $\phi_{\geq k}(S) = \mathbf{true}$ if and only if $|S| \geq k$.

We will discuss about optimization problems again in Section 7.4.1, where we will define some of their properties.

CHAPTER 4

GRAPH PARAMETERS

All results presented in Chapters 5 to 9 are directly related to some *graph parameter* and the classes containing the graphs where the value of this parameter is bounded.

Definition 4.0.1. A *graph parameter* is a function $\mathbf{p} : \mathcal{G} \to \mathbb{N}$. The *dominion* of $\mathbf{p}$ is the set of graphs where $\mathbf{p}$ is defined. We denote it by $\mathsf{dom}(\mathbf{p})$. We say that $\mathbf{p}$ is *computable* if there exists an algorithm that given a graph G, either outputs that $G \notin \mathsf{dom}(\mathbf{p})$ or outputs the value of $\mathbf{p}(G)$.

Definition 4.0.2. Given a graph parameter p and an integer $k \in \mathbb{N}$ we define the graph class $\mathcal{G}[\mathbf{p}, k]$ which contains all graphs $G \in \mathcal{G}$ that $\mathbf{p}(G) \leq k$.

To keep the notation as simple as possible, instead of $\mathsf{obs}_{\preceq}(\mathcal{G}[\mathbf{p}, k])$ we will write $\mathsf{obs}_{\preceq}(\mathbf{p}, k)$, where $\preceq$ is a partial relation on graphs, $\mathbf{p}$ a parameter and $k \in \mathbb{N}$.

Example 4.0.1. Let $\Delta : \mathcal{G} \to \mathbb{N}$ be the function where $\Delta(G) = \max \{\deg_G(u) \mid u \in V(G)\}$. This parameter is equal to the *maximum degree*

of the vertices of a graph. We can further define the graph class $\mathcal{G}[\Delta, k]$ that contains all graphs with maximum degree at most k.

Definition 4.0.3. Given a graph parameter $\mathbf{p}$ and a quasi-ordering relation $\preceq$ on graphs, we say that $\mathbf{p}$ is $\preceq$-*closed* if and only if $\mathcal{G}[\mathbf{p}, k]$ is $\preceq$-closed for every $k \in \mathbb{N}$. In other words, for two graphs H, G where $H \preceq G$, it holds that $\mathbf{p}(H) \leq \mathbf{p}(G)$.

Example 4.0.2. Δ is $\leq$, $\leq_{\text{in}}$ and $\leq_{\text{im}}$ closed but not $\leq_{\text{c}}$ and $\leq_{\text{m}}$ closed.

Some very well known – and extensively studied – graph parameters are the *minimum Vertex Cover*, the *maximum Independent Set*, the *Chromatic Number*, the *minimum Feedback Vertex Set* etc.

The vast majority of graph parameters fall too two categories: Those who can be defined using *layouts* and those that can be defined using *modifications* of the graph. In this thesis we will focus mainly on *graph layouts parameters*, and a special case of *graph modification parameters*, which has the potential to blend these two categories together (see Paragraph 4.2.1).

4.1 Layout Parameters

In this Section we attempt to familiarize the reader with the most important parameters of the first kind, i.e., those that can be defined using *layouts* in graphs. A layout is an orderings of the vertices or the edges of the graph. This type of parameters are also known as *width parameters*. As the name suggests, they are used to measure the "width" of a graph. As abstract graphs are combinatorial and not geometrical structures, the term "width" is a little misleading. In Graph Theory the width of a graph usually indicates the degree this graph "resembles" a certain family of graphs, trees or paths for instance.

One very interesting fact is that these "width" parameters are related to *Graph Searching* parameters, or *Graph Search numbers*. As a matter of fact, using graph searching games we can give an alternative – game theoretic – definition of these parameters. The main reason why there is such a connection is that both, "width" parameters and search numbers, are based on layouts.

Although our main focus is on layouts, for completion, we also give the most commonly used definitions of these parameters. More often than not, the mainstream definition is based on *graph decompositions*.

We start by making the term "layout" more precise.

Definition 4.1.1. A *vertex (edge) layout* of a graph G is a bijection $\sigma : V(G) \to [|V(G)|]$ (respectively $\sigma : E(G) \to [|E(G)|]$).

4.1.1 Treewidth

A central notion in modern Graph Theory is *treewidth*, a graph parameter which measures in what extent a graph has the topological structure of a tree ([16, 18–21, 23]). The tree-structure is very useful to set-up dynamic programming [35], a.k.a., the "sledgehammer" of the algorithms craft [185]. In many cases, we can use this tree-structure to design polynomial algorithms for problems that are NP-complete in general graphs, i.e., graphs with unbounded treewidth [19]. Therefore, small treewidth may be a basic ingredient for many important algorithmic results in Graph Theory.

There are several different definitions of this parameter, stemming from a variety of graph-theoretical notions. For instance, in Theorem 1 of [20] we can see 7 equivalent ways to define treewidth. In this thesis we will see two of the definition in [20], and an additionally definition through vertex layouts, namely the *Tree Vertex Separation number* [18, 105]. In or-

der to define tree vertex separation number, we need to fix some notation.

We denote by $\mathcal{L}_{\mathrm{V}}(G)$ the set of all vertex layouts of a graph G. Given a $\sigma \in \mathcal{L}_{\mathrm{V}}(G)$, $\sigma(u) = i$ is the *position of* $u \in V(G)$ *in* σ and $\sigma_i = \sigma^{-1}(i) = u$. For two vertices $u, v \in V(G)$, we write $u <_\sigma v$ if $\sigma(u) < \sigma(v)$. We also define

$$\sigma_{<i} = \{u \in V(G) \mid \sigma(u) < i\}$$

and, in a similar way, the sets $\sigma_{\leq i}, \sigma_{>i}$ and $\sigma_{\geq i}$.

For every $i \in [|V(G)|]$ we define the *tree-supporting set* of position i as follows:

$$S^t_\sigma(i) = \{u \in V(G) \mid \sigma(u) < i \text{ and there exists a path with } u, \sigma_i \text{ as ends, whose internal vertices belong to } \sigma_{>i}\}$$

Each layout σ is assigned with some cost. We define:

$$\mathsf{cost}_t(G, \sigma) = \max\{|S^t_\sigma(i)| \mid i \in [|V(G)|]\}$$

Definition 4.1.2 (Dendris, Kirousis, and Thilikos, 1997 [105])**.** The *tree vertex separation number* of a graph G is $\mathbf{tvs}(G) = \min\{\mathsf{cost}_t(G, \sigma) \mid \sigma \in \mathcal{L}_{\mathrm{V}}(G)\}$.

The second definition of treewidth we will see is the "classic" definition, given by Robertson and Seymour [67].

Definition 4.1.3. A *tree-decomposition* of a graph G is a pair (T, B), where T is a tree and $B : V(T) \to 2^{V(G)}$ is a function that maps every vertex $v \in V(T)$ to a subset B_v of $V(G)$ (we may refer to these sets as *bags* and to the vertices of T as *nodes*) such that:

(i) for every edge e of G there exists a vertex t in T such that $e \subseteq B_t$,

(ii) for every $v \in V(G)$, if $r, s \in V(T)$ and $v \in B_r \cap B_s$, then for every vertex t on the unique path between r and s in T, $v \in B_t$ and

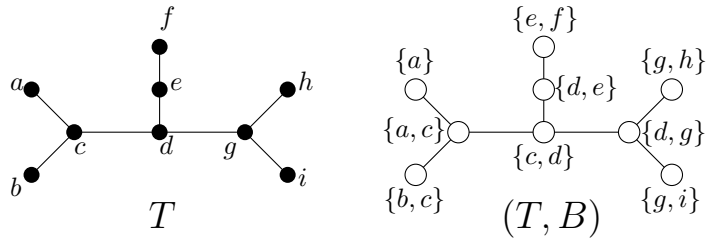


Figure 4.1: A tree-decomposition of a tree.

(iii) $\cup_{v \in V(T)} B_v = V(G)$.

The *width of a tree-decomposition* (T, B) is

$$\mathsf{width}(T, B) = \max\{|B_v| - 1 \mid v \in V(T)\}$$

and the *treewidth* of a graph G, denoted by $\mathbf{tw}(G)$, is the minimum $\mathsf{width}(T, B)$, a tree-decomposition (T, B) of G can have.

Notice the -1 on the definition of width. As our intension for treewidth is to measure the degree a graph resembles a tree, it makes sense for a tree to have treewidth one. Given a tree T, see Figure 4.1 for instance, we can define a tree-decompositions (T, B) of T (yes, we use T itself for the decomposition) as shown in Figure 4.1. In order to satisfy property (i) of the definition above, the edges of $E(T)$ must be contained in the sets B_u, $u \in V(T)$, therefore some of them must have at least two elements. Thus, we have to subtract one from the width of a decomposition in order to achieve treewidth equal to one.

Example 4.1.1. An easy exercise is to prove that $\mathbf{tw}(\boxplus_r) = r$, for every $r \geq 1$.

Tree vertex separation number and treewidth measure the same thing, as the following theorem suggests.

THEOREM 4.1.1 (Dendris, Kirousis, and Thilikos, 1997 [105])**.** *Let G be a graph. Then* $\mathbf{tvs}(G) = \mathbf{tw}(G)$.

The $\mathcal{G}[\mathbf{tw}, k]$-MEMBERSHIP problem is well known to be NP-complete [30] but, if we parametrize it by k, as **tw** is closed under taking of minors, the Graph Minors provide us with an FPT-algorithm. This algorithm has running time $O(f(k) \cdot n^3)$, for a computable function f, which can be improved, using results of the Graph Minors, to $O(f(k) \cdot n^2)$ [77]. The problem is that, we know such an algorithm exists, as the obstruction set for $\mathcal{G}[\mathbf{tw}, k]$ is finite, but we do not know a way to compute this set.

Thankfully, we have a constructive, linear time FPT-algorithm [34,37] for the parametrized problem of checking whether the treewidth of a graph is at most the size of the parameter.

THEOREM 4.1.2 (Bodlaender, 1996 [34])**.** *Let G be a graph. There exist an $O(f(k) \cdot n)$ algorithm for a computable function f, that decides if* $\mathbf{tw}(G) \leq k$ *and if so, outputs a tree-decomposition of width at most k.*

Let us see how the minor-obstruction sets for treewidth look like, for $k \leq 3$. The first two are easy to see, the third is significantly more complicated [124, 134].

OBSERVATION 4.1.1.

1. $\mathsf{obs}_{\leq_m}(\mathbf{tw}, 1) = \{K_3\}$
2. $\mathsf{obs}_{\leq_m}(\mathbf{tw}, 2) = \{K_4\}$
3. $\mathsf{obs}_{\leq_m}(\mathbf{tw}, 3) = \{K_5, K_{2,2,2}, W_8, K_2 \times C_5\}$ (see Figure 4.2)

We close this paragraph with a trivial observation, which we will use later in this thesis.

OBSERVATION 4.1.2. If G is a graph and $S = \{v \in V(G) \mid \deg_G(v) = 0\}$ then $\mathbf{tw}(G) = \mathbf{tw}(G \setminus S)$.

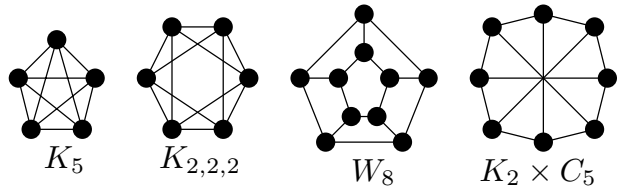


Figure 4.2: The graphs of $\mathsf{obs}_{\leq_\mathrm{m}}(\mathbf{tw}, 3)$.

4.1.2 Pathwidth

We can define *pathwidth* in a similar way as treewidth. We start with the definition that uses layouts.

Given a vertex layout σ for a graph G, we define for each $i \in [|V(G)|]$ the *path-supporting set* $S^p_\sigma(i) = N_G(\sigma_{\geq i})$, that is the set of vertices in $\sigma_{<i}$ with neighbours in $\sigma_{\geq i}$ [1], and

$$\mathsf{cost}_p(G, \sigma) = \max\{|S^t_\sigma(i)| \mid i \in [|V(G)|]\}.$$

Definition 4.1.4 (Lengauer, 1981 [114])**.** The *(path) vertex separation number* of a graph G is $\mathbf{pvs}(G) = \min\{\mathsf{cost}_p(G, \sigma) \mid \sigma \in \mathcal{L}_\mathrm{V}(G)\}$.

As the name indicates pathwidth (which we later show that is equivalent to the vertex separation number) measures the extend a graph resembles a path. To make this more obvious, we need to give a second definition of pathwidth: We will take Definition 4.1.3 and demand that T is a path. In this way we can easily define *path-decompositions* and their *width*.

Definition 4.1.5. The *pathwidth* of a graph G, $\mathbf{pw}(G)$, is the minimum $\mathsf{width}(P, B)$, where (P, B) is a path-decomposition of G.

Theorem 4.1.3 ([49, 112])**.** *Let G be a graph. Then* $\mathbf{pvs}(G) = \mathbf{pw}(G)$.

[1] Notice that we count vertices.

The definition of pathwidth was introduced in the Graph Minors series [65] and has several equivalent definitions (e.g., Theorem 2 of [20]). In Chapter 8 we define a *search game*, namely *node search*, with *search number* equivalent to pathwidth plus one.

The $\mathcal{G}[\mathbf{pw}, k]$-MEMBERSHIP problem is NP-complete [30, 48, 53, 56], but when parametrized with k, it admits a linear FPT-algorithm [34, 37] (basically it is the algorithm for treewidth).

We also must mention that pathwidth is a minor closed parameter. We present the obstruction sets for $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathbf{pw}, k)$, $k \leq 2$, in Chapter 8 (see Example 8.5.1, Theorem 8.5.4), using the node search number.

4.1.3 Cutwidth

Cutwidth, as treewidth and pathwidth, is a min-max parameter. It measures the minimum, over all vertex layouts, of the maximum number of edges between any prefix of a layout and its complement suffix. It is also closely related to the vertex separation number. Let us see why:

DEFINITION 4.1.6. Given a vertex layout σ of a graph G, the *cut at position* i, denoted by $\partial G(\sigma, i)$, is the set of crossover edges of G having one endpoint in $\sigma_{\leq i}$ and one in $\sigma_{>i}$[2]. The *width* of σ is equal to

$$\mathsf{cost}_c(G, \sigma) = \max\{|\partial_G(\sigma, i)| \mid i \in [|V(G)|]\}$$

and the *cutwidth* of G is $\mathbf{cw}(G) = \min\{\mathsf{cost}_c(G, \sigma) \mid \sigma \in \mathcal{L}_{\mathrm{V}}(G)\}$.

EXAMPLE 4.1.2. See Figure 5.1. The vertex layout $\sigma : \{a, b, c, d\} \to [4]$ with $\sigma(a) = 1$, $\sigma(b) = 2$, $\sigma(c) = 3$, and $\sigma(d) = 4$ of the graph G depicted here has $\mathsf{cost}_c(G, \sigma) = 3$.

The next observation follows immediately from the definitions.

[2]Now, we count edges.

OBSERVATION 4.1.3. If G is a graph then $\mathbf{cw}(G) \geq \mathbf{pw}(G)$.

The $\mathcal{G}[\mathbf{cw}, k]$-MEMBERSHIP is an NP-complete problem known in literature as the MINIMUM CUT LINEAR ARRANGEMENT problem [187]. From the parameterized complexity point of view, the same problem is fixed parameter tractable, as an algorithm that checks whether $\mathbf{cw}(G) \leq k$ in $f(k) \cdot n$ steps was given in [59]. Cutwidth has been extensively studied both from its combinatorial (see [39, 50, 126]) as well as its algorithmic point of view [40, 55, 58, 62].

Notice that a vertex layout is – in essence – a mono-dimensional linear arrangement of the vertices of a graph G. In Chapter 5 we will see how we can arrange the vertices of G in *Euclidean spaces* of any dimension, and the way to define the "cut" in these spaces. This will lead to a natural definition of a multi-dimensional geometric extension of cutwidth, the d-cutwidth. In Theorem 5.2.3 we will prove that this extension, and, as a consequence, cutwidth as well, is an immersion-closed parameter.

4.1.4 Linearwidth

The last layout parameter we define in this Chapter is *linearwidth*. We define some more in Chapter 8, namely the *search numbers* of *search games* and, then, one more in Chapter 9, which measures the minimum cost of an *expansion* (see Definitions 9.1.8 and 9.1.11).

Linearwidth uses edge layouts, therefore, before we give the definition, we need to review our notation and rewrite it using edges instead of vertices.

We denote by $\mathcal{L}_{\mathrm{E}}(G)$ the set of all edge layouts of a graph G. Given a $\sigma \in \mathcal{L}_{\mathrm{E}}(G)$, $\sigma(e) = i$ is the *position of* $e \in E(G)$ *in* σ and $\sigma_i = \sigma^{-1}(i) = e$. For $e_1, e_2 \in E(G)$ we will write $e_1 <_\sigma e_2$ if $\sigma(e_1) < \sigma(e_2)$. Finally, we define

$$\sigma_{<i} = \cup_{\{e \in E(G) | \sigma(e) < i\}} e$$

and accordingly the sets $\sigma_{\leq i}, \sigma_{>i}, \sigma_{\geq i}$. Notice that these are vertex sets.

Definition 4.1.7. Let G be a graph and σ an edge layout of it. We define for every $i \in [|E(G)|]$ the *line-supporting set* $S^l_\sigma(i) = \sigma_{\leq i} \cap \sigma_i$, and

$$\mathsf{cost}_l(G, \sigma) = \max\{|S^l_\sigma(i)| \mid i \in [|E(G)|]\}.$$

The *linearwidth* of a graph G is $\mathbf{lw}(G) = \min\{\mathsf{cost}_p(G, \sigma) \mid \sigma \in \mathcal{L}_{\mathrm{E}}(G)\}$ (if $|E(G)| = 1$, then $\mathbf{lw}(G) = 0$).

Linearwidth was introduced by Thomas in 1996 [61] and is closely related to the notion of *crusades*, a way to describe *mixed search*, introduced by Bienstock and Seymour in [102,120]. The $\mathcal{G}[\mathbf{lw}, k]$-Membership problem was proven to be NP-complete [120] and, in the Parameterized Complexity point of view, when parametrized by k, this problem admits a linear time FPTalgorithm. This algorithm appeared in [103] and is constructive: For any k it can construct an optimal edge arrangement with cost at most k, if one exists.

Linearwidth is also a minor closed parameter [120]. The only obstruction sets known for the graph class $\mathcal{G}[\mathbf{lw}, k]$ are for $k = 1, 2$. For $k = 1$ it consists of K_3 and the tree of Figure 8.1. For $k = 2$, its size is 52 and is given in the following theorem.

Theorem 4.1.4 (Thilikos, 2000 [120])**.** *The obstruction set* $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathbf{lw}, 2)$ *consists of the 52 graphs shown in Figure A.1 (i.e., the first figure of the appendix).*

4.2 Graph Modification Parameters

We start with two definitions.

Definition 4.2.1. A *graph property* $\mathcal{P}$ is a subset $\mathcal{P}$ of $\mathcal{G}$.

Definition 4.2.2. A *graph modification operation* is any operation that modifies a graph G to a graph G'.

These definitions are as general as a definition may possible be. Definition 4.2.2 for instance, includes not only the local transformations we saw in Section 2.3, but also includes the operations of adding edges or vertices to a graph: The *subdivision* of a graph G, where we exchange some edges of G with paths, is a graph modification operation.

Definition 4.2.3. A parameter **p** is a *graph modification parameter* (or simply a *modification parameter*) if there exists a property $\mathcal{P} \subseteq \mathcal{G}$ and a set of modification operations M, such that, for every graph G, $\mathbf{p}(G)$ is equal to the minimum number of times we have to apply an operation of M to G, in order to obtain a graph in $\mathcal{P}$.

Example 4.2.1. Let $\mathcal{P}$ be the property of "not having any edge", i.e., $\mathcal{P} = \{G = (V, E) \mid E = \emptyset\}$, and let M contain only the vertex deletion. Then, we can define the parameter **vc** which is equal to the minimum size a vertex cover of a graph may have.

Example 4.2.2. Let $\mathcal{P}$ be the property of "not having any cycles", i.e., $\mathcal{P} = \{G \mid G \text{ is a forest}\}$, and let M contain only the vertex deletion. Then, we can define the parameter **fv** which is equal to the minimum size a *feedback vertex set*[3] of a graph may have.

Example 4.2.3. Let $\mathcal{P}$ be the property of "being planar", i.e., $\mathcal{P} = \{G \mid G \text{ is planar}\}$, and let M contain only the edge deletion. Then, we can define the *"planarity"* parameter **pl** which is equal to the minimum number of edges we have to delete from a graph in order to make it planar.

The first two parameters we saw in these examples, belong to a subcategory of modification parameters, the *vertex-deletion parameters*. More examples of this category are presented in the following Section.

[3] A *feedback vertex set* of a graph is a set of vertices whose removal leaves a graph without cycles.

The modification parameters lead naturally to minimazation problems, known in literature as *Graph Modification problems* (see [144, 146, 148] for some surveys). Let **p** be a modification parameter we can define the following problem:

k-**p**-MODIFICATION	
Input:	A graph G .
Question:	Is $\mathbf{p}(G) \leq k$?

and its parameterized version:

p-**p**-MODIFICATION	
Input:	A graph G and a $k \in N$.
Parameter:	k.
Question:	Is $\mathbf{p}(G) \leq k$?

An example is the p-VERTEX COVER:

p-VERTEX COVER	
Input:	A graph G and a $k \in N$.
Parameter:	k.
Question:	Is $\mathbf{vc}(G) \leq k$?

4.2.1 Parameters defined from parameters

We will now define graph modification parameters – vertex deletion parameters to be precise – that are defined using a "host" parameter. This host parameter may well be a layout parameter (as in the case of Chapter 8).

DEFINITION 4.2.4. Let **p** be a graph parameter and $r \in \mathbb{N}$. We define the

function $(\mathbf{p}, r)$-dist : $\mathcal{G} \to \mathbb{N}$, such that

$$(\mathbf{p}, r)\text{-dist}(G) = \min\{k \mid \exists S \subseteq V(G) \text{ such that } |S| \leq k \text{ and } \mathbf{p}(G \backslash S) \leq r\}.$$

Notice that this function is also a graph parameter. It measures the number of vertices we have to delete from a graph in order for the value of **p** to drop down to r or less. Therefore, if we consider the property $\mathcal{P}_r = \{G \mid \mathbf{p}(G) \leq r\}$, we can see that $(\mathbf{p}, r)$-dist is a modification parameter. Later in this thesis we will refer to these parameters as *distance to* **p** parameters.

Example 4.2.4. Every cycle C has $(\mathbf{tw}, 1)$-dist$(C) = 1$.

Definition 4.2.5. Given a graph parameter p and two integers $r, k \in \mathbb{N}$. We define the graph class $\mathcal{G}[(\mathbf{p}, r)\text{-dist}, k]$ that contains all graphs $G \in \mathcal{G}$ for which there exists a set $S \subseteq V(G)$ of at most k vertices, such that $\mathbf{p}(G \setminus S) \leq r$.

We also define the following parameterized problem:

$(\mathbf{p}, r)$-Distance	
Input:	A graph G and a $k \in N$.
Parameter:	k.
Question:	Is $(\mathbf{p}, r)$-dist$(G) \leq k$?

The following is an immediate consequence of the definitions.

Observation 4.2.1. Let **p** be a minor closed parameter. Then for every $r \in \mathbb{N}$, $(\mathbf{p}, r)$-dist is also a minor closed parameter.

According to this observation we can include the $(\mathbf{p}, r)$-Distance problem in the following more general category of problems:

$\mathcal{F}$-Covering [4]	
Input:	A graph G and a $k \in N$.
Parameter:	k.
Question:	Does there exist $S \subseteq V(G)$ such that $\|S\| \leq k$ and $G \setminus S$ contains no graph from $\mathcal{F}$ as a minor?

where $\mathcal{F}$ is a finite set of graphs.

Notice first that, given a minor closed parameter **p** and a positive integer r, the set $\mathsf{obs}_{\leq_m}(\mathbf{p}, r)$ is finite (this of course, is due to Theorem 2.6.1). Take as $\mathcal{F}$ in the $\mathcal{F}$-Covering problem the set $\mathsf{obs}_{\leq_m}(\mathbf{p}, r)$ and observe that the $(\mathbf{p}, r)$-Distance problem is, in this case, equivalent to the $\mathsf{obs}_{\leq_m}(\mathbf{p}, r)$-Covering problem. Therefore the following holds.

Observation 4.2.2. If **p** is minor closed parameter, then for every $r \in \mathbb{N}$ the $(\mathbf{p}, r)$-Distance problem is the $\mathsf{obs}_{\leq_m}(\mathbf{p}, r)$-Covering problem.

Definition 4.2.6. Let **p** be a parameter. We say that **p** is *big in grids* if $\mathbf{p}(\boxplus_r) \to \infty$, when $r \to \infty$.

Lemma 4.2.1. Let **p** be a minor closed parameter that is big in grids. Then for every $r \in \mathbb{N}$, $\mathsf{obs}_{\leq_m}(\mathbf{p}, r)$ contains a planar graph.

Proof. Notice that, as **p** is big in grids, there exists a $c \in \mathbb{N}$ such that $\mathbf{p}(\boxplus_c) > r$. Thus, there exist an obstruction $O \in \mathsf{obs}_{\leq_m}(\mathbf{p}, r)$ such that $O \leq_m \boxplus_c$, and, consequently, O is planar. □

Definition 4.2.7. For a finite set of graphs $\mathcal{F}$, we denote by $\mathcal{G}_{\mathcal{F},k}$ the class of graphs containing all graphs G for which there exists a subset $S \subseteq V(G)$, of size at most k, such that $G \setminus S$ contains no graph from $\mathcal{F}$ as a minor.

The following proposition bounds the size of the graphs in the obstruction set for $\mathcal{G}_{\mathcal{F},k}$.

[4] These problems are also known as $\mathcal{F}$-Deletion.

Proposition 4.2.1 (Theorem 3 in [171, 172])**.** *Let $k \in \mathbb{N}$. For every finite set $\mathcal{F} \subseteq \mathcal{G}$, such that $\mathcal{F}$ contains at least one planar graph, every graph in $\mathsf{obs}_{\leq_m}(\mathcal{G}_{\mathcal{F},k})$ has size polynomial in k.*

Using Proposition 4.2.1 we can bound the size of the graphs in $\mathsf{obs}_{\leq_m}((\mathbf{p}, r)\text{-}\mathsf{dist}, k)$.

Theorem 4.2.1. *Let $r, k \in \mathbb{N}$ and let $\mathbf{p}$ be a minor closed parameter that is big in grids. Then every graph in $\mathsf{obs}_{\leq_m}((\mathbf{p}, r)\text{-}\mathsf{dist}, k)$ has size polynomial in k.*

Proof. We will prove that for every $r \in \mathbb{N}$, $\mathsf{obs}_{\leq_m}(\mathbf{p}, r)$ satisfies the properties of Proposition 4.2.1. As $\mathbf{p}$ is minor closed, from Theorem 2.6.1, we know that $\mathsf{obs}_{\leq_m}(\mathbf{p}, r)$ is finite. From Lemma 4.2.1 it holds that $\mathsf{obs}_{\leq_m}(\mathbf{p}, r)$ contains a planar graph. From Proposition 4.2.1, taking $\mathcal{F} = \mathsf{obs}_{\leq_m}(\mathbf{p}, r)$, we conclude that every graph in $\mathsf{obs}_{\leq_m}(\mathcal{G}_{\mathcal{F},k})$ has polynomial size in k. The claim holds as $\mathsf{obs}_{\leq_m}(\mathcal{G}_{\mathcal{F},k}) = \mathsf{obs}_{\leq_m}((\mathbf{p}, r)\text{-}\mathsf{dist}, k)$. □

Observation 4.2.3. Notice that Theorem 4.2.1 would also be true if, instead of asking that $\mathbf{p}$ is big in grids, we demand that for every $r \in \mathbb{N}$, $\mathsf{obs}_{\leq_m}(\mathbf{p}, r)$ contains at least one planar graph.

Proposition 4.2.2. *Let $\mathbf{p}$ be a computable parameter as in Theorem 4.2.1. If additionally this polynomial bound on the size of the graphs in $\mathsf{obs}_{\leq_m}((\mathbf{p}, r)\text{-}\mathsf{dist}, k)$, for some $k \in \mathbb{N}$, is known, then $\mathsf{obs}_{\leq_m}((\mathbf{p}, r)\text{-}\mathsf{dist}, k)$ can be computed.*

Indeed, let g be that polynomial. We can check for every graph G of size at most $g(k)$ whether $(\mathbf{p}, r)\text{-}\mathsf{dist}(G) > k$ (notice that since $\mathbf{p}$ is computable, $(\mathbf{p}, r)$-dist is also computable) and whether for every proper minor G' of G, $(\mathbf{p}, r)\text{-}\mathsf{dist}(G') \leq k$. In this way we can compute $\mathsf{obs}_{\leq_m}((\mathbf{p}, r)\text{-}\mathsf{dist}, k)$.

CHAPTER 5

D-DIMENSIONAL CUTWIDTH

Let us reconsider the definition of cutwidth and give a rough picture of what we are going to see in this Chapter. We can imagine that vertex layouts are "embeddings" of a graph in a straight line, in the following way: We assign to each vertex a unique point and each edge is given a line segment with ends the points assign to its endpoints (these line segments may contain more points assigned to vertices than their two ends)[1]. If we "cut" this line in some point not assigned to vertices, for instance point 1 in Figure 5.1, and count the line segments assigned to edges that contain this point, we can simulate the cut at a position of Definition 4.1.6. To make this clear, let us look closer at Figure 5.1. The embedding depicted corresponds to the vertex layout $\sigma : \{a, b, c, d\} \to [4]$ with $\sigma(a) = 1$, $\sigma(b) = 2$, $\sigma(c) = 3$, and $\sigma(d) = 4$ and the "cut" at the point 1 of the line corresponds to the cut at position 2.

This observation give us the motivation to consider embeddings in spaces of higher dimensions, generalizing in this way the notion of cutwidth.

[1] We will postpone the details for a while, because we need to set the framework first. However, we feel that it will be helpful for the reader to follow this train of thought.

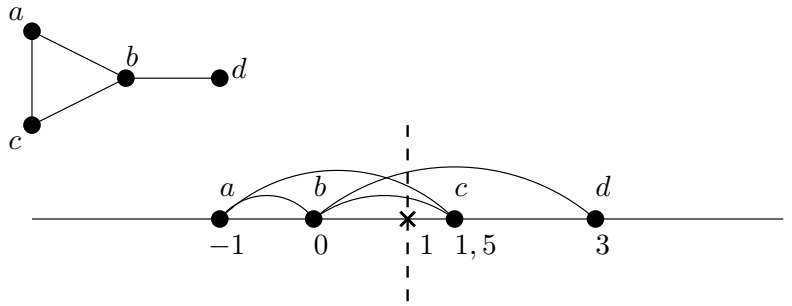


Figure 5.1: A graph and its "embedding" in a straight line (for display purposes we replace the line segments of edges with arcs).

In this Chapter we will introduce a multi-dimensional geometric extension of cutwidth, namely the *d-dimensional cutwidth* (or, simply, *d-cutwidth*) that – roughly – instead of mono-dimensional linear arrangements of a graph G, considers embeddings of G in the *d-dimensional Euclidean space* $\mathbb{R}^d$. We will define the d-cutwidth of such an embedding to be the maximum number of edges that can be intersected by a *hyperplane* of $\mathbb{R}^d$. Then, the d-cutwidth of G, denoted by $\mathbf{cw}_d(G)$, will be the minimum d-cutwidth over all such embeddings.

5.1 Graph Embeddings

We start with some definitions. To keep it short, we assume that the reader is familiar with the basic notions of *Real Analysis* and *Analytic Geometry*.

Definition 5.1.1. Every $(d-1)$-dimensional subspace Π of a d-dimensional space $\mathcal{X}$ is a *hyperplane* of $\mathcal{X}$.

Here, we are interested in hyperplanes of $\mathbb{R}^d$ (subspaces isomorphic to $\mathbb{R}^{d-1}$). Let Π be a hyperplane in $\mathbb{R}^d$, then there are $a_0, a_1, \ldots, a_d \in \mathbb{R}$ such

that $\Pi = \{(x_1, \ldots, x_d) \in \mathbb{R}^d \mid a_1x_1 + \cdots + a_dx_d + a_0 = 0\}$. We denote by $H(d)$ the set of all hyperplanes of $\mathbb{R}^d$.

DEFINITION 5.1.2. A *hypersphere*, $S(c, r)$, with *center* c and *radius* r in $\mathbb{R}^d$ is the set $\{(x_1, \ldots, x_d) \in \mathbb{R}^d \mid \sum_{i=1}^{d}(x_i - c_i)^2 = r^2\}$.

We denote by $S(d)$ the set of all hyperspheres of $\mathbb{R}^d$.

DEFINITION 5.1.3. A continuous function $C : [0, 1] \to \mathbb{R}^d$ is a *curve* of $\mathbb{R}^d$ with *ends* $C(0)$ and $C(1)$. The other points of C constitute its *interior*.

DEFINITION 5.1.4. Let $G = (V, E)$ be a graph. An *embedding* of G in the euclidean space $\mathbb{R}^d$, denoted by $\mathcal{E}_d(G)$, is a tuple $(f, \mathcal{C})$, where $f : V \to \mathbb{R}^d$ is an injection, mapping the vertices of G to $\mathbb{R}^d$ and $\mathcal{C} = \{C_e \mid e \in E\}$ is a set of curves of $\mathbb{R}^d$ with the following properties:

1. for every $e = \{u, v\} \in E$, the ends of C_e are $f(u)$ and $f(v)$
2. for all $x \in (0, 1)$ and for all $v \in V$ it holds that $C_e(x) \neq f(v)$.

For simplicity, we may sometimes refer to the elements of $f(V)$ and $\mathcal{C}$ as the vertices and edges of $\mathcal{E}_d(G)$ respectively.

DEFINITION 5.1.5. Let G be a graph. An embedding $\mathcal{E}_d(G) = (f, \mathcal{C})$, of G in $\mathbb{R}^d$, is an *essential-embedding* if for every positive integer $i \leq d$ and for every subset S of V with $|S| \geq i$, the dimension of the subspace defined by $\{f(u) \mid u \in S\}$ is $i - 1$.

We denote by $\mathbf{E}_d(G)$ the set of all essential-embeddings of G in $\mathbb{R}^d$.

For instance, an embedding of a path in $\mathbb{R}^2$ where all vertices lay on the same line is not an essential embedding. In an essential embedding of a path, every three vertices must not belong to the same line. We have to get to all this fuss about essential embeddings, because when we cut non-essential embeddings with hyperplanes we may miss some edges: If we embed the whole path in a line then every hyperplane will cross at

most one edge. This happens as, according to Definition 5.1.6, we are not allowed to cut the embedding with hyperplanes that intersect vertices. Therefore, we have to exclude non-essential embeddings from the definition of d-cutwidth.

Definition 5.1.6. Let $\mathcal{E}_d(G)$ be an essential-embedding of G in $\mathbb{R}^d$. If Π is a hyperplane of $\mathbb{R}^d$ (resp. Σ is a hypersphere of $\mathbb{R}^d$) that does not intersect any $f(v)$, $v \in V$, we denote by $\partial_G(\mathcal{E}_d(G), \Pi)$ (resp. $\partial_G(\mathcal{E}_d(G), \Sigma)$) the set of curves of $\mathcal{E}_d(G)$ that are intersected by Π (resp. Σ).

Now we are ready to define our parameter.

Definition 5.1.1. Let $G = (V, E)$ be a graph and k, d be positive integers, where $d \geq 2$. Then, we define the *d-dimensional cutwidth* of G, or simply *d-cutwidth*, to be

$$\mathbf{cw}_d(G) = \min_{\mathcal{E}_d(G) \in \mathbf{E}_d(G)} \max\{|\partial_G(\mathcal{E}_d(G), \Pi)| \mid \Pi \in H(d)\}$$

We say that an embedding $\mathcal{E}_d(G) \in \mathbf{E}_d(G)$ *realizes* d-cutwidth of G if for every hyperplane Π of $\mathbb{R}^d$, $|\partial_G(\mathcal{E}_d(G), \Pi)| \leq \mathbf{cw}_d(G)$ and the equation holds for at least one hyperplane. Observe that any hyperplane Π of $\mathbb{R}^d$ that meets a curve $C_e \in \mathcal{C}$ once (if it meets a curve more than once it is easy to see that this particular embedding does not realize the d-cutwidth of G), also meets the unique straight line segment of $\mathbb{R}^d$ with parametric equation $\sigma_e(t) = t \cdot C_e(0) + (1-t) \cdot C_e(1)$, $t \in \mathbb{R}$, i.e., the straight line segment of $\mathbb{R}^d$ that is defined by the "images" of the endpoints of edge e. Therefore, without loss of generality, we can consider only *straight-line embeddings* (see Figure 5.2).

Definition 5.1.7. Let $G = (V, E)$ be a graph. An embedding $\mathcal{E}_d(G) = (f, \mathcal{C})$, of G in $\mathbb{R}^d$, is a *straight-line embedding* if $\mathcal{C} = \{\sigma_e \mid e \in E\}$, where $\sigma_e(t) = t \cdot C_e(0) + (1-t) \cdot C_e(1)$, $t \in \mathbb{R}$ for every $C_e \in \mathcal{C}$.

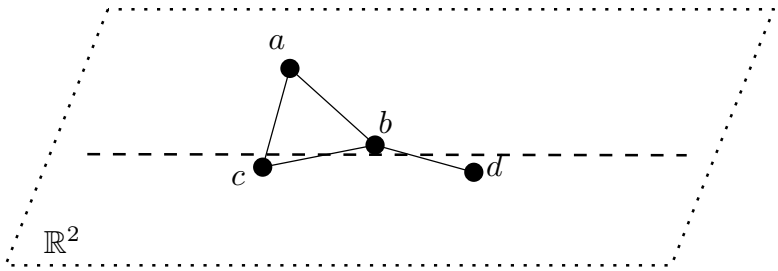


Figure 5.2: A graph with 2-cutwidth 3.

Notice that every straight line embeding $\mathcal{E}_d(G) = (f, \mathcal{C})$ is fully defined by the function f, i.e., the points assigned to the vertices, therefore, for simplicity, from now on we will omit $\mathcal{C}$.

As we described in the beginning of this Chapter, we will use the notation of Definition 5.1.1 and give an equivalent definition of cutwidth.

Definition 5.1.8. Let $G = (V, E)$ be a graph. An *embedding of G in $\mathbb{R}$*, denoted $\mathcal{E}_1(G)$, is a tuple $(f, \mathcal{I})$, where $f : V \to \mathbb{R}$ is an injection, mapping the vertices of G to $\mathbb{R}$ and $\mathcal{I} = \{(f(u), f(v)) \subset \mathbb{R} \mid \{u, v\} \in E\}$ is a set of open intervals of $\mathbb{R}$. Given an embedding $\mathcal{E}_1(G) = (f, \mathcal{I})$, we denote by $\partial_G(\mathcal{E}_1(G), x)$ the set of intervals of $\mathcal{I}$ in which x belongs.

Definition 5.1.2. Let $G = (V, E)$ be a graph and k a positive integer. We define the 1-*cutwidth* of G to be

$$\mathbf{cw}_1(G) = \min_{\mathcal{E}_1(G)} \max\{|\partial_G(\mathcal{E}_1(G), x)| \mid x \in \mathbb{R}\}.$$

It is straightforward to see that the above definition of cutwidth is equivalent to the definition given in Chapter 4.

Theorem 5.1.1. *For every graph G, $\mathbf{cw}_1(G) = \mathbf{cw}(G)$.*

Observe that, in the case of $\mathbf{cw}_1$, hyperplanes degenerate to subspaces of $\mathbb{R}$ of dimension 1 – that is points of course – and our demand of essential embeddings is expressed by our demand of injective functions. Therefore, d-cutwidth is the intuitive generalization of the notion of cutwidth in any dimension $d \geq 2$.

In the rest of this Chapter we will present some properties of d-cutwidth and discuss some algorithmic remarks.

5.2 Properties of d-cutwidth

First we have to give the following definition, as it will be used in some of the proofs of this Section.

Definition 5.2.1. Given a point $x = (x_1, \ldots, x_d)$ in $\mathbb{R}^d$ and a hyperplane $\Pi = \{(x_1, \ldots, x_d) \in \mathbb{R}^d \mid a_1x_1 + \cdots + a_dx_d + a_0 = 0\}$, the *vertical projection* of x on Π is the point $y = \{y_1, \ldots, y_d\} \in \mathbb{R}^d$ where

$$y_i = x_i - a_i \frac{a_1x_1 + \cdots + a_dx_d + a_0}{{a_1}^2 + \cdots + {a_d}^2},\ i \in \{1, \ldots, d\}.$$

The first two properties we will prove, show us that for every $d \in \mathbb{N}$, d-cutwidth is a non-trivial graph parameter.

Proposition 5.2.1. *For every graph G and every $d \geq 1$,*

$$\mathbf{cw}_d(G) \leq \mathbf{cw}_{d+1}(G).$$

Proof. Let $G = (V, E)$ be a graph, and let $\mathcal{E}_{d+1}(G) = f$ be an embedding of G in $\mathbb{R}^{d+1}$ that realizes $\mathbf{cw}_{d+1}(G)$. Let Π_0 be a hyperplane of $\mathbb{R}^{d+1}$ such that, for every $e \in E$, Π_0 is not vertical to σ_e. We vertically project $\mathcal{E}_{d+1}(G)$ on Π_0, which gives us an embedding $\mathcal{E}_d(G)$ of G in $\mathbb{R}^d$, as the restriction of f in $\mathbb{R}^d$ satisfies the conditions of a graph embedding.

Assume that there exists a hyperplane Π in $\mathbb{R}^d$ that intersects $\mathcal{E}_d(G)$ more than $\mathbf{cw}_{d+1}(G)$ times. Then we can construct a new hyperplane Π' in $\mathbb{R}^{d+1}$ that intersects $\mathcal{E}_{d+1}(G)$ more than $\mathbf{cw}_{d+1}(G)$ times. This hyperplane Π' is the hyperplane vertical to Π that passes through Π. But this fact leads to a contradiction, since any hyperplane in $\mathbb{R}^{d+1}$ can intersect $\mathcal{E}_{d+1}(G)$ at most $\mathbf{cw}_{d+1}(G)$ times. Hence, our assumption that Π' exists is false, i.e., every hyperplane in $\mathbb{R}^d$ intersects $\mathcal{E}_d(G)$ at most $\mathbf{cw}_{d+1}(G)$ times. Therefore, $\mathbf{cw}_d(G) \leq \mathbf{cw}_{d+1}(G)$. □

Proposition 5.2.2. *For every $d \geq 1$, $\mathbf{cw}_d(P_d) = d$, where by P_n we denote the path of length n.*

Proof. Let $\mathcal{E}_d(P_d)$ be an essential straight line-embedding of $P_d = (V, E)$ in $\mathbb{R}^d$, where $V = \{v_1, \ldots, v_{d+1}\}$ and $E = \{\{v_i, v_{i+1}\} \mid i \in \{1, \ldots, d-1\}\}$. Consider the midpoints m_i of $\sigma_{\{i,i+1\}}$, for every $i \in \{1, \ldots, d-1\}$. These d points of $\mathbb{R}^d$ define a hyperplane of $\mathbb{R}^d$ that intersects all edges of $\mathcal{E}_d(P_d)$. Thus $\mathbf{cw}_d(P_d) \geq d$ and as $|E(P_d)| = d$ we derive that $\mathbf{cw}_d(P_d) = d$. □

Let us now compare d-cutwidth with cutwidth and see some inequalities that hold for them.

Theorem 5.2.1. *For every graph G and every $d \geq 2$ we have:*

$$\mathbf{cw}_d(G) \leq d \cdot \mathbf{cw}(G).$$

Proof. Consider the d-dimensional curve C with parametric equation

$$C(t) := (t, t^2, t^3, \ldots, t^d), \quad t \in \mathbb{R}.$$

Consider an ordering of the nodes of G that realizes the cutwidth of G. Embed a node v_i of G to the point $p_i = C(t_i)$, for an appropriate value t_i. By "appropriate" we mean that, if a node v_i is after a node v_j in the cutwidth ordering, then the parametric value t_i corresponding to v_i

is strictly greater than the parameter value t_j corresponding to node v_j. Now embed an edge $e_{ij} = (v_i, v_j)$ of G by connecting the points p_i and p_j on C with the minimum length arc of C connecting these points.

Consider a generic hyperplane Π with equation $a_1x_1 + a_2x_2 + \ldots + a_dx_d + a_0 = 0$, where, for all i, $a_i \in \mathbb{R}$. Π can cut C at at most d points. To see that, we need to solve the system of equations:

$$\begin{aligned} a_1x_1 + a_2x_2 + \ldots + a_dx_d + a_0 &= 0, \\ x_i &= t^i, \quad i = 1, \ldots, d \end{aligned}$$

for t.

This gives the polynomial equation $q(t) := a_0 + a_1t + a_2t^2 + \ldots + a_dt^d = 0$, in t of maximum degree d. Since $q(t) = 0$ has at most d real roots, we deduce that Π intersects C at at most d points. At each point of intersection at most $\mathbf{cw}(G)$ edges of the embedding of G pass through that point. Hence, Π intersects at most $d \cdot \mathbf{cw}(G)$ edges of G, i.e., $\mathbf{cw}_d(G) \leq d\ \mathbf{cw}(G)$. □

Notice that, from Proposition 5.2.2 it holds that for every $d \geq 1$, $\mathbf{cw}_d(P_d) = d$, thus $\mathbf{cw}_d(P_d) = d \cdot \mathbf{cw}_1(P_d)$. This proves that the inequality of Theorem 5.2.1 can be tight.

COROLLARY 5.2.1. For $G = P_d$ the inequality of Theorem 5.2.1 becomes an equation.

We will now define a variant of d-cutwidth, which, instead of hyperplanes, implement hyperspheres to create the "cuts".

DEFINITION 5.2.1. Let $G = (V, E)$ be a graph and k, d be positive integers, where $d \geq 2$. Then we define the *spherical d-dimensional cutwidth* of G, or simply *spherical d-cutwidth*, to be

$$\mathbf{scw}_d(G) = \min_{\mathcal{E}_d(G) \in \mathbf{E}_d(G)} \max\{|\partial_G(\mathcal{E}_d(G), \Sigma)| \mid \Sigma \in S(d)\}$$

Let see the connection between these two parameters.

PROPOSITION 5.2.3. *For every graph G and any $d \geq 2$ we have:*

$$\mathbf{cw}_d(G) \leq \mathbf{scw}_d(G) \leq (d+1)\ \mathbf{cw}(G).$$

Proof. The left-most inequality is obvious. For every embedding $\mathcal{E}_d(G)$ of G in $\mathbb{R}^d$, the number of intersections of $\mathcal{E}_d(G)$ with a generalized hypersphere (of an appropriate center and radius) in $\mathbb{R}^d$ is greater or equal to the number of intersections of $\mathcal{E}_d(G)$ with a hyperplane in $\mathbb{R}^d$. Hence, $\mathbf{cw}_d(G) \leq \mathbf{scw}_d(G)$.

Now consider the curve $C(t) = (t, t^2, \ldots, t^d)$, with $t \in \mathbb{R}^d$, and consider the ordering of the nodes of G that realizes $\mathbf{cw}(G)$. We embed the i-th node v_i of G, in this ordering, to the point $p_i = C(i)$. We embed an edge $e_{ij} = (v_i, v_j)$ of G by connecting the points p_i and p_j on C with the minimum length arc of C connecting these points.
We claim that this curve has at most $d+1$ intersections with a generalized hypersphere S in $\mathbb{R}^d$. If S is actually a plane then we can simply apply the argumentation presented in the proof of Theorem 5.2.1. Suppose now that S is a true hypersphere, and let $\sum_{i=1}^{d}(x_i - a_i)^2 + a_0 = 0$ be the equation of S, where, for all i, $a_i \in \mathbb{R}$. Consider the following system of equations:

$$\begin{aligned} \sum_{i=1}^{d}(x_i - a_i)^2 + a_0 &= 0, \\ x_i &= t^i, \quad i = 1, \ldots, d. \end{aligned}$$

The intersections of S with the aforementioned embedding of G in C is bounded by the number of real solutions of the system above, for which t is positive. Solving this system for t we get a polynomial equation for t, namely:

$$\sum_{i=1}^{d}(t^i - a_i)^2 + a_0 = 0.$$

Expanding the above equation we get:

$$\sum_{i=1}^{d} a^{2i} - 2\sum_{i=1}^{d} a_i t^i + \sum_{i=1}^{d} a_i^2 + a_0 = 0,$$

which can be rewritten as:

$$\sum_{i=1}^{d} t^{2i} - 2\sum_{i=1}^{\lfloor \frac{d}{2} \rfloor} a_{2i} t^{2i} - 2\sum_{i=1}^{\lceil \frac{d}{2} \rceil} a_{2i-1} t^{2i-1} + \sum_{i=1}^{d} a_i^2 + a_0 = 0.$$

It is fairly easy to verify that the above equation can actually be rewritten as:

$$\sum_{i=\lfloor \frac{d}{2} \rfloor + 1}^{d} t^{2i} + \sum_{i=1}^{d} \left(\frac{1 + (-1)^i}{2} - 2a_i \right) t^i + \sum_{i=1}^{d} a_i^2 + a_0 = 0.$$

By *Descartes' rule of signs*, the number of positive real roots of this polynomial is bounded above by the number of sign variations in the sequence of its (non-zero) coefficients. Taking a close look at this polynomial, we observe that its first $d - \lfloor \frac{d}{2} \rfloor$ non-zero coefficients are equal to 1, which implies that the number of sign variations in the sequence of its coefficients is fully determined by the last $d + 2$ coefficients. A sequence of $d + 2$ real numbers can have at most $d + 1$ sign variations, hence the number of positive real roots of this polynomial is at most $d + 1$.

To finalize the proof, since any hypersphere in $\mathbb{R}^d$ intersects with C at most $d + 1$ times, we conclude that the maximum number of intersections of the embedding of G in C is at most $(d + 1)$ $\mathbf{cw}(G)$. Therefore, $\mathbf{scw}_d(G) \leq (d + 1)$ $\mathbf{cw}(G)$. □

Proposition 5.2.4. *For every graph G and every $d \geq 1$ we have:*

$$\mathbf{cw}_{d+1}(G) \leq \mathbf{scw}_d(G).$$

Proof. Consider a graph G and an embedding $\mathcal{E}_d(G)$ in $\mathbb{R}^d$ for which $\mathbf{scw}_d(G)$ is attained. Let us identify $\mathbb{R}^d$ with the hyperplane $x_{d+1} = 0$ in $\mathbb{R}^{d+1}$ and consider the unit hypersphere $\mathbb{S}^d$ in $\mathbb{R}^{d+1}$ centered at the origin. Let $\Sigma : \mathbb{R}^d \to \mathbb{S}^d$ be the stereographic projection from $\mathbb{R}^d$ to the unit hypersphere $\mathbb{S}^d$ in $\mathbb{R}^{d+1}$, and define $\mathcal{E}_{d+1}(G)$ to be the image of $\mathcal{E}_d(G)$ through the stereographic projection Σ, i.e., $\mathcal{E}_{d+1}(G) = \Sigma(\mathcal{E}_d(G))$.

Assume that there exists a hyperplane Π in $\mathbb{R}^{d+1}$ that cuts $\mathcal{E}_{d+1}(G)$ more than $\mathbf{scw}_d(G)$ times. Let S be the intersection of Π with $\mathbb{S}^d$ and let S' be the inverse image of S with respect to the stereographic projection, i.e., $S' = \Sigma^{-1}(S)$. Since S is a hypersphere laying on $\mathbb{S}^d$, S' is either a hyperplane or hypersphere in $\mathbb{R}^d$. Since the stereographic projection preserves intersections, we deduce that S' intersects $\mathcal{E}_d(G)$ more than $\mathbf{scw}_d(G)$ times. But this contradicts the definition of $\mathcal{E}_d(G)$, which implies that our assumption that Π cuts $\mathcal{E}_{d+1}(G)$ more than $\mathbf{scw}_d(G)$ times is false. Hence, we found an embedding of G in $\mathbb{R}^{d+1}$, for which the maximum number of intersections with any hyperplane in $\mathbb{R}^{d+1}$ is at most $\mathbf{scw}_d(G)$. Therefore, $\mathbf{cw}_{d+1}(G) \leq \mathbf{scw}_d(G)$. □

Let us now push the inequality of Theorem 5.2.1 a little further.

Theorem 5.2.2. *For any graph G,*

$$\mathbf{cw}_3(G) \leq 2\ \mathbf{cw}_2(G).$$

Proof. Let G be a graph and consider an ordering of the nodes of G that realizes $\mathbf{cw}(G)$.

Consider an axis-aligned ellipse E centered at the origin with its x-axis being greater than its y-axis (e.g., the ellipse $4x^2 + y^2 - 4 = 0$). Embed the nodes of G on the positive half of E, denoted as $E_{1/2}$, that is on the half-ellipse that lies on the positive halfplane with respect to the x-axis, in such a way so that their x-coordinates preserve the ordering. In other words, given two nodes v_i and v_j of G, such that v_i precedes v_j in

the node ordering, then $x_i < x_j$. Let us call $\mathcal{E}_2(G)$ the above-mentioned embedding.

Given a generalized circle C on the plane, C can cut the half-ellipse $E_{1/2}$ at most two times. Hence we found an embedding of G in $\mathbb{R}^2$ such that any generalized circle C in $\mathbb{R}^2$ intersects this embedding at most $2 \cdot \mathbf{cw}(G)$ times. In other words, $\mathbf{scw}_2(G) \leq 2\ \mathbf{cw}(G)$. Using the results from Propositions 5.2.1 and 5.2.4 we conclude that:

$$\mathbf{cw}_3(G) \leq \mathbf{scw}_2(G) \leq 2\ \mathbf{cw}(G) \leq 2\ \mathbf{cw}_2(G),$$

which is what we wanted to prove. □

Cutwidth is not a minor closed parameter but it is immersion closed [22]. We will prove that the same holds for d-cutwidth.

THEOREM 5.2.3. *Let $G = (V, E)$ be a graph and H be an immersion (strong immersion) of G. For every $d \geq 1$, $\mathbf{cw}_d(H) \leq \mathbf{cw}_d(G)$.*

Proof. Let $\mathcal{E}_d(G)$ be an an embedding of G in $\mathbb{R}^d$ that realizes $\mathbf{cw}_d(G)$. Let H' be a subgraph of G. Then, given $\mathcal{E}_d(G) = (f, \mathcal{C})$, we define an embedding $\mathcal{E}_d(H') = (f_{H'}, \mathcal{C}_{H'})$ of H' in $\mathbb{R}^d$, where $f_{H'}$ is the restriction of f to $V(H')$ and $\mathcal{C}_{H'} = \{c_e \in \mathcal{C} \mid e \in E(H')\}$. Observe that every hyperplane of $\mathbb{R}^d$ that intersected $l \leq \mathbf{cw}_d(G)$ edges of $\mathcal{E}_d(G)$ intersects at most l edges of $\mathcal{E}_d(H')$, therefore $\mathbf{cw}_d(H) \leq \mathbf{cw}_d(G)$.

We will next prove that if H is the result of one lift of edges $e_1 = \{u, v\}$ and $e_2 = \{v, w\}$ of E, then $\mathbf{cw}_d(H) \leq \mathbf{cw}_d(G)$. Clearly, for any hyperplane R of $\mathbb{R}^d$ its corresponding numbers $|\partial_H(\mathcal{E}_d(H), R)|$ and $|\partial_G(\mathcal{E}_d(G), R)|$ can differentiate only due to the intersections of R with edges e, e_1 and e_2. Let $E(H) \ni e = \{u, w\}$ be the resulting edge of the lift. Let $\Pi \in H(d)$ be a hyperplane that does not intersect $f(v)$, $\forall v \in V$. If Π intersects e (more accurately, the straight line segment that represents e in the embedding in $\mathcal{E}_d(G)$), then Π separates $\mathbb{R}^d$ into two halfspaces (i.e., subspaces of dimension d), namely A and B. Assume, without loss of

generality, that $u \in A$ and $w \in B$. Then, as Π does not intersect $f(v)$, either $v \in A$ or $v \in B$, which means that either Π intersects e_1 or Π intersects e_2 in $\mathcal{E}_d(G)$. Therefore, $|\partial_H(\mathcal{E}_d(H), \Pi)| \leq |\partial_G(\mathcal{E}_d(G), \Pi)|$. The same holds trivially for the case that Π does not intersect e. Therefore, $\mathbf{cw}_d(H) \leq \mathbf{cw}_d(G)$. Summing up the above we get that, for every graph H that is the result of some (or maybe none) vertex deletions, edge deletions and edge lifts of G, $\mathbf{cw}_d(H) \leq \mathbf{cw}_d(G)$, which is what we stated. Notice that the above proof implies the same relation between a graph G and a strong immersion H of G. □

5.3 Algorithmic Remarks About d-cutwidth

As a consequence of the Nash-Williams Conjecture's proof in [87], for every positive integer k, the class of weak immersion minimal graphs with d-cutwidth bigger than k contains a finite set of graphs. Notice that this class is exactly $\mathsf{obs}_{\leq_{\text{im}}}(\mathbf{cw}_d, k)$, i.e., the *immersion obstruction set* for d-cutwidth at most k. This fact, combined with Theorem 5.2.3, implies that $\mathbf{cw}_d(G) \leq k$ if and only if none of the graphs in $\mathsf{obs}_{\leq_{\text{im}}}(\mathbf{cw}_d, k)$ is contained in G as an immersion, thus, the graphs of $\mathsf{obs}_{\leq_{\text{im}}}(\mathbf{cw}_d, k)$ characterize graphs with d-cutwidth at most k.

Moreover, according to Theorem 3.2.2, checking whether an n-vertex graph contains as an immersion some k-vertex graph H, can be done in $f(k)\cdot n^3$ steps, for some computable function f. As a consequence, checking whether $\mathbf{cw}_d(G) \leq k$ can be done in $|\mathsf{obs}_{\leq_{\text{im}}}(\mathbf{cw}_d, k)| \cdot f(k) \cdot n^3$ steps[2]. This running time can become linear (on n) using the inequality of Theorem 5.2.1.

Proposition 5.3.1. *For every k and d, there exists a linear (on n) time algorithm that, given a graph G, decides whether $\mathbf{cw}_d(G) \leq k$ or not.*

[2] To be precise, the $f(k)$ factor in the running time of this algorithm is in fact $f(n(O))$ where O is the graph in $\mathsf{obs}_{\leq_{\text{im}}}(\mathbf{cw}_d, k)$ having the most vertices.

Indeed, the algorithm first checks whether $\mathbf{cw}(G) \leq k$, using the linear time algorithm in [59]. If the answer is negative, then from Lemma 5.2.1 we can safely report that $\mathbf{cw}_d(G) > k$. If not, then, as it holds that $\mathbf{pw}(G) \leq \mathbf{cw}(G)$ (see e.g. [59]), G has a path decomposition with width at most k and , according to [37] such a decomposition can be found in linear time. Therefore, we can then check whether some of the graphs in $\mathsf{obs}_{\leq_{\mathsf{im}}}(\mathbf{cw}_d, k)$ is contained in G as an immersion in $f(k) \cdot n$ steps, using dynamic programming for instance.

Unfortunately, the algorithm above is non-constructive as we have no other knowledge about the set $\mathsf{obs}_{\leq_{\mathsf{im}}}(\mathbf{cw}_d, k)$, except from the fact that it is finite. Whether we can obtain a *constructive* $f(k) \cdot n$ steps algorithm for d-cutwidth remains an open problem.

Another way to prove Proposition 5.3.1 is to express d-cutwidth in MSO, observe that for every graph G it holds that

$$\mathbf{tw}(G) \leq \mathbf{pw}(G) \leq \mathbf{cw}(G) \leq \mathbf{cw}_d(G)$$

(this follows from the definition of treewidth and pathwidth, Observation 4.1.3, and Proposition 5.2.1), and use *Courcelle's Theorem*:

Theorem 5.3.1 (Courcelle, 1990 [189])**.** *Every graph property definable in MSO of graphs can be decided in linear time on graphs of bounded treewidth.*

This algorithm goes as follows:

Step 1: Use the algorithm of Theorem 4.1.2 and check whether the input graph G has $\mathbf{tw}(G) > k$. If so, return No and **stop**.

Step 2: (As $\mathbf{tw}(G) \leq k$ we can implement Theorem 5.3.1.) Run the algorithm of Theorem 5.3.1 and check if $\mathbf{cw}_d(G) \leq k$.

The problem with this approach is that in order to express d-cutwidth in MSO we have to define embeddings in MSO, which in term involves discretizing $\mathbb{R}^d$.

5.4 Conclusion

What we did in this Chapter was to define d-cutwidth, a multi-dimensional geometric extension of cutwidth and then proved some of its properties. Theorem 5.4.1 states the most important of them.

Theorem 5.4.1. *The following hold:*
i. *d-cutwidth is immersion closed.*
ii. *For every graph G and every $d \geq 1$, $\mathbf{cw}_d(G) \leq \mathbf{cw}_{d+1}(G)$.*
iii. *For every graph G and every $d \geq 1$, $\mathbf{cw}_d(G) \leq d \cdot \mathbf{cw}(G)$.*
iv. *For every graph G, $\mathbf{cw}_3(G) \leq 2 \cdot \mathbf{cw}_2(G)$.*
v. *For every k and d, there exists a linear (on n) time algorithm that, given a graph G, decides whether $\mathbf{cw}_d(G) \leq k$ or not.*

Some interesting open problems and suggestions for future work about this parameter are the following:

- Is the problem of checking for a graph G whether $\mathbf{cw}_d(G) \leq k$ NP-complete? This seems to be very difficult as – in a sense – we have to find the right way to discretize $\mathbb{R}^d$ in order to define equivalent classes of embeddings and make their number finite. If we do not accomplish that, there is no hope of finding certificates of polynomial size.

- Finding a constructive algorithm for checking whether $\mathbf{cw}_d(G) \leq k$, for $d \geq 2$, as we discussed in the previous Section, is an insisting open problem.

- It would be very interesting if we extended Theorem 5.4.1 (iv) and proved that $\mathbf{cw}_{d+1}(G) \leq f(d) \cdot \mathbf{cw}_d(G)$, for some function f.

- We used hyperplanes and hyperspheres to "cut" embeddings of graphs. We are very curious to find out what happens when we use other geometrical objects for this purpose.

CHAPTER 6

OBSTRUCTIONS FOR UNIONS OF CLASSES

Let us take a moment to briefly recapitulate what we have discussed so far. In Chapter 2 we presented the Graph Minors Theory. We argued that it constitutes a vital part of modern Combinatorics, as many theorems that were proven – and techniques that were introduced – in its context, are of great significance, not only in Algorithmics and the theory of Parameterized Complexity, but in *Structural Graph Theory* as well. The most prominent results in Graph Minors Theory, supporting this point, are the *Excluded Grid Theorem* [92], the Structural Theorems in [77, 80] and the *Irrelevant Vertex Technique* in [86]. Some further examples of algorithmic applications, can be found in [165, 179].

While the minor partial ordering has been extensively studied throughout the last decades [77, 80, 85–87, 92, 123, 125], the immersion ordering has only recently gained more attention [42, 137, 179]. In Chapter 2 we saw one of the fundamental results concerning this relation, the proof of Nash-Williams' Conjecture (the weak version). As Corollary 2.6.1 indicates, a graph class $\mathcal{C}$ that is closed under taking of immersions, can be characterized by a finite family $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C})$ of forbidden immersions.

Some examples of graph classes for which such characterizations are possible, are the class $\mathcal{E}_t$ of graphs that admit a *(proper) edge-coloring* that uses at most $t \in \mathbb{N}$ colors such that for every two edges of the same color every path between them contains an edge of greater color (for more information, see [36]) and the class $\mathcal{W}_k$ of graphs whose *carving-width* is at most k, $k \in \mathbb{N}$, (for more information see [151]). Another example – we plan to study extensively – is the classes $\mathcal{G}[\mathbf{cw}_d, k]$ for $d \geq 2$ and $k \geq 0$ (see Chapter 5).

Furthermore, in Theorem 2.6.3, we saw that there exists an algorithm – with running time $O(|V(G)|^3)$ – that decides whether a graph H is an immersion of a graph G (where the hidden constants depend only on H). Combining this algorithm with the forbidden immersions characterization one can prove that it can be decided – in cubic time – whether a graph belongs to $\mathcal{C}$ or not.

This – meta-algorithmic – result for an immersion-closed graph class $\mathcal{C}$ assumes that the family $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C})$ is known. Of course, the finiteness of $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C})$ directly implies that this family of graphs is computable. However, an algorithm for the computation of $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C})$ would also require a description of the family. Therefore, while the *existence* of an algorithm computing the immersion obstruction set of an immersion-closed graph class $\mathcal{C}$ is affirmed, the *construction* of such an algorithm is, in general, elusive.

To make matters even worse, recall that the proofs of Theorem 2.6.1 and Conjecture 2.6.1 are non-constructive (see [88]). This means that this proofs do not provide us with a generic algorithm that would allows us to identify the obstruction sets for every minor or immersion-closed graph class. Even when we work with fixed graph classes, this task can be extremely challenging, as such a set may contain a huge number of graphs [129] and no general upper bound on its cardinality is known other than its finiteness.

6.1 Computation of Obstruction Sets

The issue of the computability of obstruction sets for minors and immersions was raised by Fellows and Langston [192, 193] and the challenges against computing obstruction sets soon became clear. In particular, in [193] Fellows and Langston showed that the problem of determining obstruction sets from machine descriptions of minor-closed graph classes is *recursively unsolvable*[1] (which directly holds for the immersion ordering as well). Moreover, in [127] Courcelle, Downey and Fellows proved that the obstruction set of a minor-closed graph class cannot be computed when only a description of the minor-closed graph class in MSO is known. Thus, the following is a natural open problem:

> *Identify the information needed in order to make it possible to compute the obstruction set* $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C})$ *of an immersion-closed graph class* $\mathcal{C}$.

In other words, our goal is to look for theorems able to delimit the computability horizon of obstruction sets for the immersion relation.

Several methods have been proposed towards tackling the non-constructiveness of these sets (see, for example, [125, 192]) and the problem of algorithmically identifying minor obstruction sets has been extensively studied [52, 123, 125, 127, 192, 193]. In [125], it was proven that the obstruction set of a minor-closed graph class $\mathcal{C}$ which is the union of two minor-closed graph classes $\mathcal{C}_1$ and $\mathcal{C}_2$ whose obstruction sets are given can be computed under the assumption that there is at least one tree that does not belong to $\mathcal{C}_1 \cap \mathcal{C}_2$, and in [123] it was shown that the aforementioned assumption is not really necessary. This also lead to an interesting question:

[1] I.e., there does not exist a Turing Machine that takes as input the descriptions of a minor-closed graph class $\mathcal{C}$ and outputs $\mathsf{obs}_{\leq_{\text{m}}}(\mathcal{C})$.

What kind of Set Theory operations $\square$ on two immersion closed graph classes $\mathcal{C}_1$ and $\mathcal{C}_2$, for which the sets $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C}_1)$ and $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C}_2)$ are known, make the computation of $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C}_1 \square \mathcal{C}_2)$ possible[2]?

In this Chapter we deal with the problem of computing $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C})$ for families of graph classes $\mathcal{C}$ that are constructed by finite unions of immersion-closed graph classes. Observe that the union and the intersection of two immersion-closed graph classes are also immersion-closed, hence their obstruction sets are of finite size. It is also easy to see that, given the obstruction sets of two immersion-closed graph classes, the obstruction set of their intersection can be computed in a trivial way (it is just the union of the two immersion obstruction sets).

However, in general, while the combination of a *machine description* of an immersion-closed graph class $\mathcal{C}$, that is, an algorithm deciding the membership of a graph in $\mathcal{C}$, combined with an upper bound on the size of the obstructions makes the computation of $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C})$ possible, neither the machine description of the class nor the upper bound alone are sufficient information. Moreover, as mentioned before, no generic procedure is known for computing such an upper bound. Thus, the problem of computing the obstruction set of the union of two immersion-closed graph classes is not trivial at all.

Our final goal for this Chapter is to prove that there exist an algorithm that, given the obstruction sets of two immersion-closed graph classes, outputs the obstruction set of their union. Our approach is based on the derivation of a uniform upper bound on the tree-width of the subgraph-minimal graphs that do not belong to an immersion-closed graph class $\mathcal{C}$. We will build on the machinery introduced by *Isolde Adler, Martin Grohe* and *Stephan Kreutzer* in [123] for computing minor obstruction sets. To

[2] Provided, of course, that $\mathcal{C}_1 \square \mathcal{C}_2$ is also immersion-closed.

be more precise, the algorithm we are going to present needs an MSO-description of an immersion-closed graph class $\mathcal{C}$ (instead of a machine description) and an upper bound on the tree-width of the subgraph-minimal graphs that contain an obstruction of $\mathcal{C}$ (instead of an upper bound on the size of the obstructions of $\mathcal{C}$).

To fulfil this goal, we will adapt the results on [123] so as to permit the computation of the obstruction set of any immersion-closed graph class $\mathcal{C}$, under the aforementioned conditions. We present this algorithm in Lemma 6.2.2, with which we conclude the computability part of this Chapter.

The next step is to give a combinatorial result proving a uniform upper bound on the tree-width of the subgraph-minimal graphs that do not belong to the union of two immersion-closed graph classes, whose obstruction sets are known. These combinatorial proofs make use of a suitable extension of the *Unique Linkage Theorem* of *Ken-ichi Kawarabayashi* and *Paul Wollan* [89], which, in turn, is an improvement of the *Vital Linkage Theorem* of Robertson and Seymour in [85, 86]. In particular, notice that a subgraph-minimal graph G not belonging to an immersion-closed graph class $\mathcal{C}$ contains an immersion obstruction for $\mathcal{C}$, that is, there is a set of *terminals* that are joined by edge-disjoint paths[3] which make use of all the vertices and edges of G. As G is subgraph-minimal, this implies that these edge-disjoint paths of G correspond to vertex-disjoint paths in its line graph which also form a *unique linkage* (both the definition of Linkages and the Unique Linkage Theorem will be presented in Section 6.3). Then, the Unique Linkage Theorem, allows us to prove a bound on the tree-width of the line graph of G and consequently, to the tree-width of G.

The reader is advised to have this proof sketch in mind when go into the details.

[3] Remember that we are always referring to weak immersions.

6.2 Computing Immersion Obstruction Sets

As we mentioned before, to make everything work we need the following combinatorial lemma.

LEMMA 6.2.1. There exists a computable function $f : \mathbb{N} \to \mathbb{N}$ such that the following holds. Let H and G be graphs such that $H \leq_{\text{im}} G$. If G' is a minimal subgraph of G with $H \leq_{\text{im}} G'$ then $\mathbf{tw}(G') \leq f(|E(H)|)$.

We are not going into the details of the the proof of Lemma 6.2.1, as we will later prove a stronger statement of it (Lemma 6.3.3). Instead we will continue by giving the necessary definitions in order to prove the analogue of Lemma 3.1 in [123] for the immersion ordering. We first give more formal definitions of the contraction, minor, and immersion relation.

DEFINITION 6.2.1. Let H and G be two graphs. H is a *contraction* of G if there exists a surjection $\phi : V(G) \to V(H)$ such that:

(1) for every vertex $v \in V(H)$, $G[\phi^{-1}(v)]$ is connected, and

(2) for every two distinct vertices $u, v \in V(G)$, it holds that $\{v, u\} \in E(G)$ if and only if the graph $H[\phi^{-1}(v) \cup \phi^{-1}(u)]$ is connected.

If furthermore H is not isomorphic to G, we say that H is a *proper contraction* of G.

Observe that function ϕ indicates the edges of $E(G)$ we will contract in order to obtain H. There are the edges of $G[\phi^{-1}(v)]$, for $v \in V(H)$.

DEFINITION 6.2.2. Let H and G be two graphs. H is a *minor* of G if there is a function $\psi : V(H) \to 2^{V(G)}$ such that:

(1) for every $v \in V(H)$, $B_v = G[\psi(v)]$ is connected, and

(2) for any two distinct vertices v, w of H, B_v and B_w share no common vertex, and for every edge $\{u, v\} \in E(H)$, there is an edge in G with one endpoint in B_v and one in B_u.

The graph obtained by the union of all B_v, $v \in V(H)$, and the edges between B_v and B_u in G, where $\{v, u\} \in E(H)$, is called a *minor model of* H *in* G. This graph is obtained from G after the deletion of some vertices and edges. We can further contract the edges of $G[B_v]$, $v \in V(H)$, to obtain H as a minor of G. A model with minimal number of vertices and edges is called *minimal minor model*.[4]

In Section 7.5.4 we will give another – more general – definition of the minor relation.

Definition 6.2.3. Let H and G be two graphs. H is an *immersion* of G if there exists an injective function $\psi : V(H) \to V(G)$, such that:

(1) for every edge $\{u, v\} \in E(H)$, there is a path from $\psi(u)$ to $\psi(v)$ in G, and

(2) for any two distinct edges of H the corresponding paths in G are edge-disjoint.

Moreover, if these paths are vertex disjoint from $\psi(V(H))$, then we say that H is a *strong immersion* of G. The function ψ indicates the ends of the paths we will lift in order to obtain H as an immersion of G. It is called an *immersion model of* H *in* G and a model with minimal number of vertices and edges is called *minimal immersion model*.

To facilitate the proofs in this Chapter we have to establish an ordering between finite sets of graphs.

[4]Notice that we could define the *minor* relation for the two graphs by removing in the second property of Definition 6.2.1 the demand that if $\{u, v\} \notin E(H)$ then $G[\phi^{-1}(v) \cup \phi^{-1}(u)]$ is not connected, but we chose not to.

Definition 6.2.4. For two finite sets of graphs, say $\mathcal{F}_1$ and $\mathcal{F}_2$, we will write $\mathcal{F}_1 \leq \mathcal{F}_2$ if and only if:

1. $\sum_{G \in \mathcal{F}_1} |V(G)| < \sum_{H \in \mathcal{F}_2} |V(H)|$ or
2. $\sum_{G \in \mathcal{F}_1} |V(G)| = \sum_{H \in \mathcal{F}_2} |V(H)|$ and $\sum_{G \in \mathcal{F}_1} |E(G)| \leq \sum_{H \in \mathcal{F}_2} |E(H)|$.

We now state a theorem which plays a crucial role in the proof of the algorithm for the computation of immersion obstructions for general immersion-closed graph classes.

Theorem 6.2.1 (Arnborg, Lagergren, and Seese, 1991 [31])**.** *For every positive integer k, it is decidable given an MSO-formula whether it is satisfied by a graph G whose tree-width is upper bounded by k.*

In [123], Adler, Grohe and Kreutzer provide the tools that will allow us to use Theorem 6.2.1 to compute the obstruction sets of minor-closed graph classes, when an upper bound on the tree-width of the subgraph-minimal graphs not belonging to a minor-closed graph class is known and an MSO-description of the graph class can be computed.

In this Chapter we will adapt the machinery of [123] to the immersion ordering and provide a generic technique to compute immersion obstruction sets when an explicit value of such upper bound on the tree-width is known.

Moreover, in Section 6.3 we will obtain such a – computable – upper bound on the tree-width for the case where $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ and $\mathcal{C}_1$, $\mathcal{C}_2$ are immersion-closed graph classes whose obstruction sets are given, and therefore show that $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C})$ can be computed.

6.2.1 An extension of MSO

For convenience, we consider the extension of the signature $\tau_{\mathcal{G}}$ to a signature τ_{ex} that pairs the representation of a graph G with the representation

of one of its tree-decompositions.

Definition 6.2.5. If G is a graph and $\mathcal{T} = (T, B)$ is a tree-decomposition of G, τ_{ex} is the signature that consists of the relation symbols V, E, I of $\tau_{\mathcal{G}}$, and four more relation symbols V_T, E_T, I_T and B.
A *tree-dec-expansion* of G and $\mathcal{T}$, is a τ_{ex}-structure

$$\mathfrak{G}_{ex} = (V(G) \cup E(G) \cup V(T) \cup E(T), \\ V^{\mathfrak{G}_{ex}}, E^{\mathfrak{G}_{ex}}, I^{\mathfrak{G}_{ex}}, V_T^{\mathfrak{G}_{ex}}, E_T^{\mathfrak{G}_{ex}}, I_T^{\mathfrak{G}_{ex}}, B^{\mathfrak{G}_{ex}})$$

where $V_T^{\mathfrak{G}_{ex}} = V(T)$ represents the node set of T, $E_T^{\mathfrak{G}_{ex}} = E(T)$ the edge set of T, $I_T^{\mathfrak{G}_{ex}} = \{(v, e) \mid v \in e \cap V(T) \wedge e \in E(T)\}$ the incidence relation in T and $B^{\mathfrak{G}_{ex}} = \{(t, v) \mid t \in V(T) \wedge v \in B_t \cap V(G)\}$.

Let $\mathcal{T}_k$ be the class of graphs of tree-width at most k. We denote by $\mathcal{C}_{\mathcal{T}_k}$ the class of tree-dec expansions consisting of a graph G with $\mathbf{tw}(G) \leq k$, and a tree decomposition (T, B) of G of $\mathsf{width}(T, B) \leq k$.

Proposition 6.2.1 (Adler, Grohe, and Kreutzer, 2008 [123])**.**

1. *Let G be a graph and (T, B) a tree decomposition of it with* width *$(T, B) \leq k$. Then the tree-width of the tree-dec expansion of G is at most $k + 2$.*

2. *There is an MSO-sentence $\phi_{\mathcal{C}_{\mathcal{T}_k}}$ such that for every τ_{ex}-structure $\mathfrak{G}$, $\mathfrak{G} \models \phi_{\mathcal{C}_{\mathcal{T}_k}}$ if and only if $\mathfrak{G} \in \mathcal{C}_{\mathcal{T}_k}$.*

Notice that by Theorem 6.2.1, for every $k \geq 0$, it is possible to decide if an MSO-formula is satisfied in a graph G of $\mathbf{tw}(G) \leq k$. An immediate corollary of this result and Proposition 6.2.1 is the following.

Corollary 6.2.1. It can be decided, for every k, if an MSO-formula ϕ is satisfied in some $\mathfrak{G} \in \mathcal{C}_{\mathcal{T}_k}$.

THEOREM 6.2.2 (Adler, Grohe, and Kreutzer, 2008 [123]). *For every $k \geq 0$, there is an MSO-sentence $\phi_{\mathcal{T}_k}$ such that for every tree-dec expansion $\mathfrak{G} \in \mathcal{C}_{\mathcal{T}_l}$ of G, for some $l \geq k$, it holds that $\mathfrak{G} \models \phi_{\mathcal{T}_k}$ if and only if $\mathbf{tw}(G) = k$.*

The algorithm of Lemma 6.2.2 can be applied not only to MSO definable graph classes, but also to *Layer-wise MSO definable classes*:

DEFINITION 6.2.6. A graph class $\mathcal{C}$ is *layer-wise MSO-definable*, if for every $k \in \mathbb{N}$ we can compute an MSO-formula ϕ_k such that $G \in \mathcal{C}$ and $\mathbf{tw}(G) \leq k$ if and only if $\mathfrak{G} \models \phi_k$, where $\mathfrak{G} \in \mathcal{C}_{\mathcal{T}_k}$ is a tree-dec expansion of G.

DEFINITION 6.2.7. Let $\mathcal{C}$ be an immersion-closed graph class. The *width* of $\mathcal{C}$, denoted by $\mathsf{width}(\mathcal{C})$, is the minimum positive integer k such that for every graph $G \notin \mathcal{C}$ there is a graph $G' \leq G$ with $G' \notin \mathcal{C}$ and $\mathbf{tw}(G') \leq k$.

We have to stress here that Lemma 6.2.1 ensures that the width of an immersion-closed graph class is well-defined.

OBSERVATION 6.2.1. If $\mathcal{C}_1$ is an immersion-closed graph class then the following holds. For every graph $G \notin \mathcal{C}_1$, there exists a graph $G' \leq G$ such that $G' \notin \mathcal{C}_1$ and $\mathbf{tw}(G') \leq \max\{f(|E(H)|) \mid H \in \mathsf{obs}_{\leq_{\mathrm{im}}}(\mathcal{C}_1)\}$, where f is the function of Lemma 6.2.1.

We are now able to present an algorithm analogue to that of Lemma 3.1 in [123], but in our case for the immersion ordering.

LEMMA 6.2.2. There exists an algorithm that, given an upper bound $l \geq 0$ on the width of a layer-wise MSO-definable class $\mathcal{C}$, and a computable function $f : \mathbb{N} \to \text{MSO}$ such that for every positive integer k, $f(k) = \phi_k$, where ϕ_k is the MSO-formula defining $\mathcal{C} \cap \mathcal{T}_k$, it computes $\mathsf{obs}_{\leq_{\mathrm{im}}}(\mathcal{C})$.

Proof. In order to prove the Lemma it is enough to prove the following.

Claim 1. For any finite family of graphs $\mathcal{F} = \{F_1, \ldots, F_n\}$, it is decidable whether the following two conditions are unsatisfiable for any graph G.

$$G \in \mathcal{C} \text{ and there exists an } F \in \mathcal{F} \text{ such that } F \leq_{\text{im}} G \tag{6.1}$$

$$G \notin \mathcal{C} \text{ and for every } F \in \mathcal{F}, F \not\leq_{\text{im}} G \tag{6.2}$$

To see that the above claim is enough, first notice that if $\mathcal{F}$ is a finite family of graphs for which the conditions (6.1) and (6.2) are unsatisfiable then $\mathcal{F}$ is a forbidden immersion characterization of $\mathcal{C}$, that is, a graph G belongs to $\mathcal{C}$ if and only if it does not contain any of the graphs in $\mathcal{F}$ as an immersion. By definition, $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C})$ is the minimum such family according to the $\leq$-relation (Definition 6.2.4).

Thus, if Claim 1 holds, we can find the set $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C})$ by enumerating, according to $\leq$ (see Definition 6.2.4), all the finite families of graphs $\mathcal{F}$ and deciding, for each one of them, if the conditions (6.1) and (6.2) are unsatisfiable.

Proof of Claim 1. Let G be a graph in $\mathcal{C}$ such that $F \leq_{\text{im}} G$, for some $F \in \mathcal{F}$. Lemma 6.2.1 implies that there exists a graph $G' \leq G$ such that $\mathbf{tw}(G') \leq f(|E(F)|)$ and $F \leq_{\text{im}} G'$, where f is the function of Lemma 6.2.1. Observe that $G' \in \mathcal{C}$. Thus, (6.1) is satisfiable if and only if there exists a graph in $\mathcal{C}$, whose tree-width is bounded from $\max\{f(|E(F)|) \mid F \in \mathcal{F}\}$, that satisfies it, where f is the computable function of Lemma 6.2.1. Let $\phi_{\mathcal{C}}$ be the formula defining $\mathcal{C} \cap \mathcal{T}_k$ in $\mathcal{C}_{\mathcal{T}_k}$, and $\phi_{\mathcal{F}} = \bigvee_{F \in \mathcal{F}} \phi_F$, where ϕ_F is the formula from Lemma 2.7.1 and $k = \max\{f(|E(F)|) \mid F \in \mathcal{F}\}$. Notice that there exists some graph $G \in \mathcal{C}$ that models $\phi_{\mathcal{F}}$ if and only if $\phi_{\mathcal{C}} \wedge \phi_{\mathcal{F}}$ is satisfiable for some $G' \in \mathcal{C}_{\mathcal{T}_k}$. From Corollary 6.2.1, this is decidable.

Let $G \notin \mathcal{C}$ be a graph such that $F \not\leq_{im} G$, for every $F \in \mathcal{F}$. Recall that the width of a graph class $\mathcal{C}$ is the minimum positive integer k such that for every graph $G \notin \mathcal{C}$ there is a $G' \leq G$ with $G' \notin \mathcal{C}$ and $\mathbf{tw}(G') \leq k$. Thus, G contains a subgraph G' with tree-width at most w such that $G' \notin \mathcal{C}$,

where w is computable by Lemma 6.2.1. Observe that $F \not\leq_{im} G'$, for every $F \in \mathcal{F}$. If $\phi'_{\mathcal{C}}$ is the MSO-sentence defining $\mathcal{C} \cap \mathcal{T}_w$ (given by the hypothesis), then there exists a graph $G \notin \mathcal{C}$ such that $F \not\leq_{im} G$, for every $F \in \mathcal{F}$ if and only if $\neg\phi'_{\mathcal{C}} \wedge \neg\phi_{\mathcal{F}}$ is satisfiable in $\mathcal{C}_{\mathcal{T}_w}$. The decidability of whether $\neg\phi'_{\mathcal{C}} \wedge \neg\phi_{\mathcal{F}}$ is satisfiable in $\mathcal{C}_{\mathcal{T}_w}$ follows, again, from Corollary 6.2.1. □

As Claim 1 holds, the lemma follows. □

Corollary 6.2.2. There is an algorithm that given an MSO formula ϕ and $k \in \mathbb{N}$, so that ϕ defines an immersion closed-graph class $\mathcal{C}$ of width at most k, computes the obstruction set of $\mathcal{C}$.

Notice that a direct proof of Corollary 6.2.2 would be enough for our goal for this Chapter, which is to prove Theorem 6.3.2 . However, as Lemma 6.2.2 provides a more general framework regarding the constructiveness of immersion obstruction sets, it is worth to be mentioned.

Although Lemma 6.2.2 provides an algorithm for computing the obstruction set of any immersion-closed graph class $\mathcal{C}$, given that the conditions stated are satisfied, this result is generic and there is no uniform way for computing either an upper bound on the width of $\mathcal{C}$ or an MSO-description of $\mathcal{C}$.

In the next Section, we will prove some combinatorial lemmata and then conclude that if $\mathcal{C}_1$ and $\mathcal{C}_2$ are two immersion-closed graph classes whose obstruction sets are known then the width of $\mathcal{C}_1 \cup \mathcal{C}_2$ can be computed and hence, the set $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C}_1 \cup \mathcal{C}_2)$ is computable.

6.3 Width Bounds for Immersion-closed Graph Classes

Our goal for this Section is to give an upper bound on the width of the graph class $\mathcal{C}_1 \cup \mathcal{C}_2$, where $\mathcal{C}_1$ and $\mathcal{C}_2$ are immersion-closed graph classes

and their obstruction sets are known. This will directly imply the proof of Lemma 6.2.1. To do this, we will first prove a generalization of the Unique Linkage Theorem, appeared in the Graph Minors series [85]. Then we will introduce the notion of an *r-approximate edge-linkage* and work on the subgraph-minimal graphs not belonging to $\mathcal{C}_1 \cup \mathcal{C}_2$.

Finally, as it is trivial to compute an MSO-description of $\mathcal{C}_1 \cup \mathcal{C}_2$ when we are given the sets $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C}_1)$ and $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C}_2)$, we will show that the obstruction set of $\mathcal{C}_1 \cup \mathcal{C}_2$ is computable.

6.3.1 Linkages

We start with a series of definitions.

Definition 6.3.1. Let r be a positive integer. An *r-approximate linkage* in a graph G is a family L of paths with distinct endpoints in G such that for every $r+1$ distinct paths $P_1, P_2, \ldots, P_{r+1}$ in L, it holds that $\bigcap_{i \in [r+1]} V(P_i) = \emptyset$. These paths are the *components* of the linkage.

Definition 6.3.2. Let $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ and $(\beta_1, \beta_2, \ldots, \beta_k)$ be elements of $V(G)^k$, that is k-tuples of vertices of $V(G)$. An r-approximate linkage L, consisting of the paths $P_1, P_2, \ldots, P_k$, *links* $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ and $(\beta_1, \beta_2, \ldots, \beta_k)$ if P_i is a path with endpoints α_i and β_i, for every $i \in [k]$.

Definition 6.3.3. Given a r-approximate linkage L for a graph G, we define the *order* of L to be equal to the number of its paths. We call an r-approximate linkage of order k, *r-approximate k-linkage*. When $r = 1$, such a family of paths is simply called *linkage*.

Definition 6.3.4. Two r-approximate k-linkages L and L' are *equivalent* if for every component P of L there exists a component P' of L' with the same endpoints. An r-approximate linkage L of a graph G is called *unique* if for every linkage L' that is equivalent to L, $V(L) = V(L')$.

Definition 6.3.5. A linkage L in a graph G is called *vital* if there is no other linkage in G joining the same pairs of vertices.

In [85], Robertson and Seymour proved a theorem which is known as *The Vital Linkage Theorem*. This theorem provides an upper bound for the tree-width of a graph G that contains a vital k-linkage L such that $V(L) = V(G)$, where the bound depends only on k. A stronger statement of the Vital Linkage Theorem was recently proved by Kawarabayashi and Wollan [89], where instead of asking for the linkage to be vital, it asks for it to be unique. Notice here that a vital linkage is also unique. As in some of the proofs in this Chapter (for instance, the proof of Lemma 6.3.1) we deal with unique but not necessarily vital linkages we make use of the Vital Linkage Theorem in its latter form stated below.

Theorem 6.3.1 (Unique Linkage Theorem, 2010 [89])**.** *There exists a computable function $w : \mathbb{N} \to \mathbb{N}$ such that the following holds: Let L be a (1-approximate) k-linkage in G with $V(L) = V(G)$. If L is unique then* $\mathbf{tw}(G) \leq w(k)$.

Lemma 6.3.1. There exists a computable function $f : \mathbb{N} \to \mathbb{N}$ such that the following holds. Let G be a graph that contains a 2-approximate k-linkage $\tilde{L}$ such that $V(\tilde{L}) = V(G)$. If $\tilde{L}$ is unique, then $\mathbf{tw}(G) \leq f(k)$.

Proof. Let G be a graph that contains a unique 2-approximate k-linkage $\tilde{L}$ with $V(\tilde{L}) = V(G)$ that links $A = (\alpha_1, \alpha_2, \ldots, \alpha_k)$ and $B = (\beta_1, \beta_2, \ldots, \beta_k)$ in G. Denote by T the set $A \cup B$ and consider the graph G^b with

$$\begin{aligned} V(G^b) &= V((G \setminus T) \times K_2) \cup T \\ E(G^b) &= E((G \setminus T) \times K_2) \cup \{\{t, t'\} \mid t, t' \in T \wedge \{t, t'\} \in E(G)\} \\ &\quad \cup \{\{t, (v, x)\} \mid t \in T \wedge x \in V(K_2) \wedge v \in V(G) \\ &\quad \wedge \{t, v\} \in E(G)\}, \end{aligned}$$

where $V(K_2) = \{1, 2\}$. It is easy to see that G^b contains a k-linkage that links A and B. Let G' be a minimal induced subgraph of G^b that contains

a k-linkage L' that links A and B. From Theorem 6.3.1, it follows that

$$\mathbf{tw}(G') \leq w(k). \tag{6.3}$$

From now on we work towards proving that $G \leq_{\mathrm{m}} G'$. In order to achieve this, we prove the following two claims for G'.

Claim 2. If L' is a k-linkage in G' that links A and B then for every vertex $v \in V(G) \setminus T$ no path of L' contains both $(v, 1)$ and $(v, 2)$.

Proof of Claim2. Towards a contradiction, assume that for some vertex $v \in V(G) \setminus T$, there exists a (t, t')-path P of L' that contains both $(v, 1)$ and $(v, 2)$. Without loss of generality, assume also that $(v, 1)$ appears before $(v, 2)$ in P. Let y be the successor of $(v, 2)$ in P and notice that $y \neq (v, 1)$. From the definition of G^b and the fact that G' is an induced subgraph of G^b, $\{y, (v, 1)\} \in E(G') \setminus E(L')$. By replacing the subpath of P from $(v, 1)$ to y with the edge $\{(v, 1), y\}$, we obtain a linkage in $G' \setminus (v, 2)$ that links A and B. This contradicts to the minimality of G'. □

Claim 3. If L' is a k-linkage in G' that links A and B then for every vertex $v \in V(G) \setminus T$, $V(L') \cap \{(v, 1), (v, 2)\} \neq \emptyset$.

Proof of Claim3. Assume, in contrary, that there exists a linkage L' in G' and a vertex $x \in V(G) \setminus T$ such that L' links A and B and $V(L') \cap \{(x, 1), (x, 2)\} = \emptyset$. Claim 2 ensures that, after contracting the edges $\{(v, 1), (v, 2)\}$, $v \in V(G) \setminus T$ (whenever they exist), the corresponding paths compose a 2-approximate k-linkage $\tilde{L}'$ of $G \setminus \{x\}$ that links A and B. This is a contradiction to the assumption that $\tilde{L}$ is unique. Thus, the claim holds. □

Recall that $T \subseteq V(G')$ and that G' is an induced subgraph of G^b. Claim 3 implies that we may obtain G from G' by contracting the edges $\{(v, 1), (v, 2)\}$ for every $v \in V(G) \setminus T$ (whenever they exist). As $G \leq_{\mathrm{m}} G'$, from (6.3), it follows that, $\mathbf{tw}(G) \leq w(k)$. □

We remark that, the previous lemma holds for any graph G that contains an r-approximate k-linkage. This can be seen by substituting $(G \setminus T) \times K_2$ with $(G \setminus T) \times K_r$ in its proof.

The following lemma provides an upper bound on the tree-width of a graph G, given an upper bound on the tree-width of its line graph $L(G)$.

Lemma 6.3.2. If G is a graph and k is a positive integer with $\mathbf{tw}(L(G)) \leq k$ then $\mathbf{tw}(G) \leq 2k + 1$.

Proof. Notice that, from Observation 4.1.2, we may assume that G does not contain isolated vertices. Suppose that G is graph such that $L(G)$ admits a tree decomposition of width at most k and recall that every vertex of $L(G)$ corresponds to an edge of G. We construct a tree decomposition $\mathcal{T} = (T, B)$ of G from a tree decomposition $\mathcal{T}_L$ of $L(G)$ by replacing in each bag of $\mathcal{T}_L$ every vertex of $L(G)$ by the endpoints of the corresponding edge in G. It is easy to verify that this is a tree decomposition of G. Therefore, $\mathbf{tw}(G) \leq 2k + 1$. □

Before we proceed to the next lemma, we need to introduce the notion of an *r-approximate k-edge-linkage* in a graph.

Definition 6.3.6. An *r-approximate edge-linkage* in a graph G is a family of paths E in G such that for every $r+1$ distinct paths $P_1, P_2, \ldots, P_{r+1}$ in E, it holds that $\cap_{i \in [r+1]} E(P_i) = \emptyset$. We call these paths the *components* of the edge-linkage.

Definition 6.3.7. Let $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ and $(\beta_1, \beta_2, \ldots, \beta_k)$ be elements of $V(G)^k$. We say that an r-approximate edge-linkage E, consisting of the paths $P_1, P_2, \ldots, P_k$, *links* $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ and $(\beta_1, \beta_2, \ldots, \beta_k)$ if P_i is a path with endpoints α_i and β_i, for every $i \in [k]$.

Definition 6.3.8. Given a r-approximate edge-linkage E for a graph G, we define the *order* of E to be equal to the number of its paths. We call

an r-approximate edge-linkage of order k, *r-approximate k-edge-linkage*. When $r = 1$, we call such a family of paths, an *edge-linkage*.

Lemma 6.3.3. There exists a computable function r such that the following holds. Let G_1, G_2 and G be graphs such that $G_i \leq_{\text{im}} G$, $i = 1, 2$. If G' is a minimal subgraph of G where $G_i \leq_{\text{im}} G'$, $i = 1, 2$, then

$$\mathbf{tw}(G') \leq r(|E(G_1)|, |E(G_2)|).$$

Proof. Let G' be a minimal subgraph of G such that $G_i \leq_{\text{im}} G'$, $i = 1, 2$. Recall that the edges of G_i compose a k_i-edge-linkage E_i in G, where $k_i = |E(G_i)|$, $i = 1, 2$. Furthermore, observe that the paths of E_1 and E_2 constitute a 2-approximate k-edge-linkage E of G, where $k = k_1 + k_2$. Indeed, notice that in contrary to linkages, we do not require the paths that are forming edge-linkages to have different endpoints. The minimality of G' implies that $\bigcup\{P \mid P \in E\} = G'$. Denote by $A = (v_{i_1}, v_{i_2}, \ldots, v_{i_k})$ and $B = (v_{j_1}, v_{j_2}, \ldots, v_{j_k})$ the vertex sets that are edge-linked by E in G' and let $\widehat{G}$ be the graph with

$$\begin{aligned} V(\widehat{G}) &= V(G') \cup \{u_{i_q} \mid q \in [k]\} \cup \{u_{j_q} \mid q \in [k]\}, \\ E(\widehat{G}) &= E(G') \cup \{t_{i_q} \mid q \in [k]\} \cup \{t_{j_q} \mid q \in [k]\}, \end{aligned}$$

where the vertices u_{i_q} and u_{j_q}, $q \in [k]$ are new, $t_{i_q} = \{u_{i_q}, v_{i_q}\}$, $q \in [k]$ and $t_{j_q} = \{u_{j_q}, v_{j_q}\}$, $q \in [k]$.

Consider the line graph of $\widehat{G}$, $L(\widehat{G})$, and notice that E corresponds to a 2-approximate k-linkage L from A_L to B_L in $L(\widehat{G})$, where $A_L = (t_{i_1}, t_{i_2}, \ldots, t_{i_k})$ and $B_L = (t_{j_1}, t_{j_2}, \ldots, t_{j_k})$. This is true as, from the construction of $\widehat{G}$, all the vertices in A_L and B_L are distinct. The minimality of G' yields that $V(L) = V(L(\widehat{G}))$ and implies that L is unique. From Lemma 6.3.1, we obtain that $\mathbf{tw}(L(\widehat{G})) \leq f(k)$. Therefore, from Lemma 6.3.2, we get that $\mathbf{tw}(\widehat{G}) \leq p(f(k))$, where p is the function of Lemma 6.3.2. Finally, as $G' \leq \widehat{G}$, $\mathbf{tw}(G') \leq r(k_1, k_2)$, where $r(k_1, k_2) =$

$p(f(k_1 + k_2))$. □

Notice that Lemma 6.2.1 follows from Lemma 6.3.3 when we set G_2 to be the empty graph.

To conclude with the necessary conditions for Lemma 6.2.2 we have to show that given two immersion-closed graph classes $\mathcal{C}_1$ and $\mathcal{C}_2$ the immersion-closed graph class $\mathcal{C}_1 \cup \mathcal{C}_2$ is layer-wise MSO-definable.

Observation 6.3.1. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be immersion-closed graph classes, then $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ is a layer-wise MSO-definable class defined, for every $k \geq 0$, by the formula

$$\phi_k = \left(\left(\bigwedge_{G \in \mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C}_1)} \neg \phi_G \right) \vee \left(\bigwedge_{H \in \mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C}_2)} \neg \phi_H \right) \right) \wedge \phi_{\mathcal{C}_{\mathcal{T}_k}}$$

where ϕ_G and ϕ_H is the formula described in the proof of Lemma 2.7.1, and $\phi_{\mathcal{C}_{\mathcal{T}_k}}$ the formula of Proposition 6.2.1.

We are now able to prove the main result of this Section.

Theorem 6.3.2. *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two immersion-closed graph classes. If the sets $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C}_1)$ and $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C}_2)$ are given, then the set $\mathsf{obs}_{\leq_{\text{im}}}(\mathcal{C}_1 \cup \mathcal{C}_2)$ is computable.*

Proof. Observation 6.3.1, provides us with an MSO-description of the immersion-closed graph class $\mathcal{C}_1 \cup \mathcal{C}_2$, and Lemma 6.3.3 gives us an upper bound on the width of $\mathcal{C}_1 \cup \mathcal{C}_2$. Therefore, Lemma 6.2.2 is applicable. □

6.4 Conclusion

In Chapter 2 we mentioned the fact that Robertson and Seymour claimed that the class of graphs is also well-quasi-ordered under the *strong* immersion ordering [87]. However, a full proof of this result has not appeared

so far. The combinatorial results we discussed in this chapter, namely, the upper bounds on the width of a graph class, also hold for the strong immersion ordering. Thus, if the claim of Robertson and Seymour is true and eventually proven, the obstruction set of the union of two strongly immersion-closed graph classes, whose obstruction sets are given, can also be computed.

Finally, it was proven by Courcelle, Downey and Fellows [127] that the obstruction set of a minor-closed graph class $\mathcal{C}$ cannot be computed by an algorithm whose input is a description of $\mathcal{C}$ as an MSO-formula. When we consider the immersion relation, the computability of the obstruction set of an immersion-closed graph class $\mathcal{C}$, given solely an MSO description of $\mathcal{C}$, remains an open problem.

CHAPTER 7

MONOTONE KERNELS

In this Chapter we introduce the concepts of *parameter-invariant* and *minor-monotone* kernels for optimization problems. Our approach (being consistent with our main focus in this thesis) will be "parameter-centered", meaning that our central Theorem (Theorem 7.3.1) is stated using parameters instead of problems. In order for it to be applied to an optimization problem p-Π, and prove the existence of a linear kernel, we use the parameter $\mathbf{p}_{p\text{-}\Pi}$ that measures the size of the optimal solution for p-Π (see Subsection 7.4.1). Theorem 7.3.1 demands that the parameter – in addition to being computable and minor-closed – satisfies the condition of being *Protrusion decomposable* and of having *Finite Integer Index*. These extra conditions, if met from the parameter $\mathbf{p}_{p\text{-}\Pi}$, imply the existence of linear kernels (for p-Π) that have the desirable kernel properties we plan to introduce in this Chapter.

The first concerns the *monotonicty* of a kernel for a graph problem, with respect to some ordering relation on graphs (see Definition 3.3.2). A typical example of an $\leq_{\text{in}}$-*monotone* (or *induced subgraph-monotone*) kernel is the size $2k$ kernel for the p-Vertex Cover problem in [157].

Also, induced subgraph-monotone kernels have been defined and studied in [150]. For our purposes here, $\preceq$ in Definition 3.3.2 will be the minor relation, and, thus, in addition to Conditions (a) and (b) in Definition 3.3.1, we demand that the following condition is satisfied:

(c) $G' \leq_{\mathrm{m}} G$.

I.e., we demand that the graph in the returned instance is a minor of the graph of the input instance and we restrict our attention to minor-closed problems on graphs. We should mention that the notion of *minor-monotonicity* of kernels appeared also in [183], but for kernels of exponential size.

The second kernel property is the *parameter invariance* property of Definition 3.3.3:

(d) $k' = k$.

Here we demand that the parameter of the input instance remains invariant. This condition applies to the general definition of kernelization and does not require that the parameterized problem in question is a graph problem.

Parameter-invariant kernels should not be confused with *proper kernels*, defined in [150], where it is demanded that $k' \leq k$, i.e., the parameter only decreases in the new equivalent instance. This property is much weaker than parameter invariance [1]. A slightly more general version of kernelization is *strict kernelization*, defined in [170] which demands that $k' \leq k + c$ for some constant c. Actually it is mentioned in [150]/ [170] that a parameterized problem admits an FPT algorithm if and only if it admits a proper/strict kernelization. However, as we will see, enforcing parameter-invariance needs much more effort.

[1] It follows from the results in [155] that all linear kernels emerging from the application of Proposition 7.4.1 in page 119 – "a master Theorem for linear kernels" – are proper.

A similar notion to parameter invariant-kernels, and – in a sense – more general concept, is the one of α-*approximate kernels* where, for every $c \geq 1$, a c-approximate solution to the preprocessed instance can be turned, in polynomial time, into a $c\alpha$-approximate solution to the original instance. α-approximate kernels have been recently introduced in the soon-to-be-seminal work of [181] and established an important link between kernelization and approximation algorithms. Under this vewpoint, parameter-invariant kernels can be seen as α-approximate kernels where $\alpha = 1$.

In this chapter we will present some of the consequences kernels satisfying (c) and (d) have. The first (presented in Section 7.4.4) is of combinatorial nature: These kernels make the computation of obstruction sets possible, for minor closed parameters, whose value drops in a constant factor when we do a local transformation to a graph, such as a vertex/edge deletion or an edge contraction (we can say informally that they "behave" in the normal – intuitive – way, see Definition 7.4.9 for more details).

The second is algorithmic. The existence of such kernels accelerates the running time of known EPTAS for several graph optimization problems from $f(1/\epsilon) \cdot n^{O(1)}$ to $n^{O(1)} + f(1/\epsilon) \cdot OPT^{O(1)}$ (see Section 7.4.5 for more details).

Because of the "size" of this Chapter, and the fact that it contains many and diverse notions, we will divide it into small steps that eventually lead to the proof of Theorem 7.3.1 – our final goal.

The first milestone is to define *protrusion decompositions* and the FII *property*, in order to have the notation set and be able to state this theorem.

7.1 Protrusions

Definition 7.1.1. Let G be a graph and let $R \subseteq V(G)$. We say that R is a β-*protrusion* of G if $\max\{|\partial_G(R)|, \mathbf{tw}(G[R])\} \leq \beta$. An (α, β)-*protrusion*

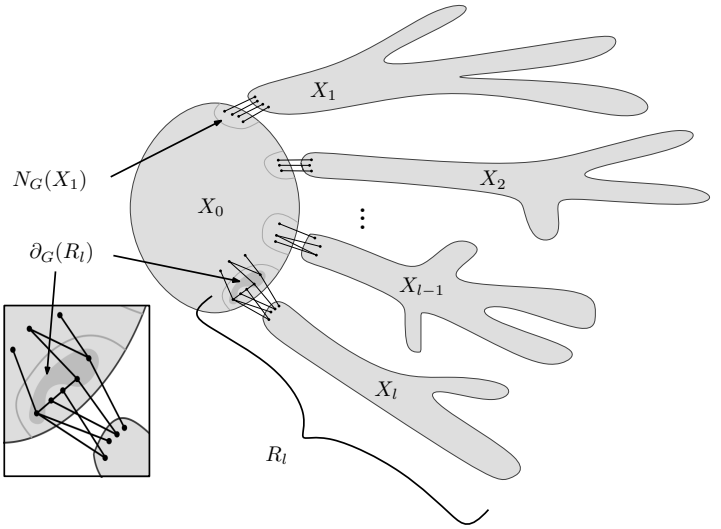


Figure 7.1: Notice that for every $i \in [l]$ the set $\partial_G(R_i)$ contains the vertices of R_i that are incident to vertices of X_0 and that $\partial_G(R_i) \subseteq N_G(X_i)$.

decomposition of a graph G is a partition $\mathcal{P} = \{X_0, X_1, \ldots, X_\ell\}$ of $V(G)$ such that

(1) $\max\{\ell, |X_0|\} \leq \alpha$,

(2) for every $i \in [\ell]$, the set $R_i = N_G[X_i]$ is a β-protrusion of G and

(3) for every $i \in \{1, \ldots, \ell\},\ N_G(X_i) \subseteq X_0$.

We call the sets R_i, $i \in [\ell]$, the *protrusions* of $\mathcal{P}$ and the set X_0 the *core* of $\mathcal{P}$ (see Figure 7.1).

Definition 7.1.2. We say that a parameter $\mathbf{p}$ is *protrusion decomposable*, if there exists some $c > 0$ such that:

$$\forall G \in \mathsf{dom}(\mathbf{p}),\ G \text{ has a } (c \cdot \mathbf{p}(G), c)\text{-protrusion decomposition} \quad (7.1)$$

We denote by dec($\mathbf{p}$) the minimum c for which (7.1) is true and we call it *protrusion decomposability constant* of $\mathbf{p}$.

7.2 Boundaried Gaphs and FII

Definition 7.2.1. A *labeling* of a graph G is any injective function $\lambda : V(G) \to \mathbb{N}$.

Definition 7.2.2. Let $t \in \mathbb{N}$. A *t-boundaried graph* is a triple $\mathbf{G} = (G, X, \lambda)$ where G is a graph, $X \subseteq V(G)$, $|X| = t$, and λ is a labelling of G. We call X the *boundary* of $\mathbf{G}$ and we call the vertices of X the *boundary vertices* of $\mathbf{G}$. We also call G the *underlying graph* of $\mathbf{G}$ and the integer $t = |X|$ the *boundary size* of $\mathbf{G}$.

Definition 7.2.3. We say that $\mathbf{G}$ is a *boundaried graph* if there exists a positive integer t such that $\mathbf{G}$ is a t-boundaried graph.

We denote $V(\mathbf{G}) = V(G)$, $E(\mathbf{G}) = E(G)$, and $|\mathbf{G}| = |G|$.

Definition 7.2.4. We define $\mathcal{B}^{(t)}$ as the set of all t-boundaried graphs and we set $\mathcal{B}^{(\leq t)} = \bigcup_{t' \in \{0\} \cup [t]} \mathcal{B}^{(t')}$.

We also define $\mathcal{T}^{(\leq t)}$ to be the set of all boundaried graphs in $\mathcal{B}^{(\leq t)}$ whose underlying graph has treewidth at most $t - 1$.

Definition 7.2.5. Given a t-boundaried graph $\mathbf{G} = (G, X, \lambda)$, we define the *label normalising function* of $\mathbf{G}$, $\psi_{\mathbf{G}} : X \to [t]$ such that for each $v \in X$,

$$\psi_{\mathbf{G}}(v) = |\{u \in X \mid \lambda(u) \leq \lambda(v)\}|.$$

Note that, as λ is an injective function, $\psi_{\mathbf{G}}$ is a bijection. Given a boundary vertex v of $\mathbf{G}$, we call $\psi_{\mathbf{G}}(v)$ the *index* of v.

Definition 7.2.6. We define the *frontier graph* $H_{\mathbf{G}}$ of $\mathbf{G}$ as follows:

$$H_{\mathbf{G}} = ([t], (\{\psi_{\mathbf{G}_q}(x), \psi_{\mathbf{G}}(y)\} \mid \{x, y\} \in E(G[X])).$$

We say that two t-boundaried graphs $\mathbf{G}_1$ and $\mathbf{G}_2$ are *compatible* if $H_{\mathbf{G}_1} = H_{\mathbf{G}_2}$ (not just isomorphic).

Definition 7.2.7. Let $\mathbf{G}_1 = (G_1, X_1, \lambda_1)$ and $\mathbf{G}_2 = (G_2, X_2, \lambda_2)$ be two t-boundaried graphs. We define the *gluing operation* $\oplus$ such that

$$(G_1, X_1, \lambda_1) \oplus (G_2, X_2, \lambda_2)$$

is the graph G obtained by taking the disjoint union of G_1 and G_2 and then, for each $i \in [t]$, identifying the vertex $\psi_{\mathbf{G}_1}^{-1}(i)$ and the vertex $\psi_{\mathbf{G}_2}^{-1}(i)$ (i.e., we identify boundary vertices of the same index).

Keep in mind that $\mathbf{G}_1 \oplus \mathbf{G}_2$ is a graph and not a boundaried graph. Moreover, the operation $\oplus$ requires both boundaried graphs to have boundaries of the same size.

Definition 7.2.8. Let $\mathbf{p}$ be a graph parameter and $t \in \mathbb{N}$. Let $\mathbf{G}_i = (G_i, X_i, \lambda_i) \in \mathcal{B}^{(\leq t)}, i \in [2]$. We say that $\mathbf{G}_1 \equiv_{\mathbf{p},t} \mathbf{G}_2$ if $\mathbf{G}_1$ and $\mathbf{G}_2$ are both t'-boundaried graphs for some $t' \in \{0\} \cup [t]$, they are compatible, and

$$\exists c_{\mathbf{G}_1,\mathbf{G}_2} \in \mathbb{Z} \ \ \forall \mathbf{F} \in \mathcal{B}^{(t')}, \quad \mathbf{p}(\mathbf{G}_1 \oplus \mathbf{F}) = \mathbf{p}(\mathbf{G}_2 \oplus \mathbf{F}) + c_{\mathbf{G}_1,\mathbf{G}_2} \tag{7.2}$$

If in the above definition for some $i \in [2]$ the graph $\mathbf{G}_i \oplus \mathbf{F} \notin \mathsf{dom}(\mathbf{p})$, we assume that $\mathbf{p}(\mathbf{G}_i \oplus \mathbf{F}) = \infty$.

It is easy to observe that $\equiv_{\mathbf{p},t}$ is an equivalence relation on the set $\mathcal{B}^{(\leq t)}$. Moreover, one of the equivalence classes of this relation is the set, denoted $\mathsf{null}(\mathbf{p})$, of all boundaried graphs whose underlying graph does not belong in $\mathsf{dom}(\mathbf{p})$.

Definition 7.2.9. We say that a graph class $\mathcal{R}$ is a *t-representative collection* for $\equiv_{\mathbf{p},t}$ if it contains one member of each of its equivalence classes, except from $\mathsf{null}(\mathbf{p})$. Given a $\mathbf{G} \in \mathcal{B}^{(\leq t)}$ we define $\mathsf{rep}_{\mathbf{p}}(\mathbf{G})$ to be the (unique) boundaried graph $\mathbf{G}'$ in $\mathcal{R}$ such that $\mathbf{G}' \equiv_{\mathbf{p},t} \mathbf{G}$.

Definition 7.2.10. We say that $\mathbf{p}$ has *Finite Integer Index* (FII) if $\equiv_{\mathbf{p},t}$ has a finite set of equivalence classes for every t, i.e., the number of its equivalence classes of $\equiv_{\mathbf{p},t}$, depends on $\mathbf{p}$ and t only.

Definition 7.2.11. Given that $\mathbf{p}$ has FII, we define $\mathsf{card}_{\mathbf{p}}(t)$ to be the number of equivalence classes of $\equiv_{\mathbf{p},t}$ for each $t \in \mathbb{N}$.

7.3 Main Theorem

To specify the dependencies of the constants hidden inside the *Big O* we introduce the following notation.

Definition 7.3.1. Let $p, q \in \mathbb{N}$ with $p \geq 2$ and $0 < q < p$, let $f : \mathbb{N}^p \to \mathbb{N}$, and let $g : \mathbb{N}^q \to \mathbb{N}$. We say that

$$f(x_1, \ldots, x_p) = O_{x_q,\ldots,x_p}(g(x_1, \ldots, x_q))$$

if there is a function $h : \mathbb{N}^{p-q} \to \mathbb{N}$ such that

$$f(x_1, \ldots, x_p) = O(h(x_q, \ldots, x_p) \cdot g(x_1, \ldots, x_{p-1})).$$

We can now state our main goal for this Chapter (this is our second milestone). We plan to prove the following:

Theorem 7.3.1. *For every graph parameter* $\mathbf{p}$ *that has* FII*, is computable, protrusion decomposable, and minor-closed, there is a constant* $c_{\mathbf{p}}$ *and a polynomial algorithm that given a graph* $G \in \mathsf{dom}(\mathbf{p})$*, outputs a graph* G' *such that*

1. $G' \leq_{\mathrm{m}} G$*,*
2. $\mathbf{p}(G') = \mathbf{p}(G)$*, and*
3. $|G'| \leq c_{\mathbf{p}} \cdot \mathbf{p}(G)$.

Moreover, this algorithm runs in $O_{c_{\mathbf{p}}}(|G|^{2c+2})$ *steps where* $c = \mathsf{dec}(\mathbf{p})$ *is the protrusion decomposability constant of* $\mathbf{p}$.

The constant $c_{\mathbf{p}}$ is defined in Section 7.5.5, and is a combination of many functions.

Theorem 7.3.1 proves the existence of such an algorithm, but is not constructive, i.e., it cannot provide us with an algorithm that takes as input a parameter $\mathbf{p}$, satisfying its specification, and outputs an algorithm computing the constant $c_{\mathbf{p}}$ and the graph G'. The problem is that the algorithm Theorem 7.3.1 provide us with, needs to know a representative collection for $\equiv_{\mathbf{p},t}$, or at least $\mathsf{card}_{\mathbf{p}}$ must be a computable function (see Lemma 7.7.1).

The importance of Theorem 7.3.1 is suggested from its applications presented in the next Section.

7.4 Consequences of Theorem 7.3.1

In order to reach our third milestone for this Chapter, which is to present some of the consequences of Theorem 7.3.1, we have to get a little deeper into optimization problems.

7.4.1 More on optimization problems

Let us define some more properties an optimization problem may have. We already mentioned the function $\mathbf{p}_{p\text{-}\Pi}$ which give us the value of the parameter in the optimal solution.

Definition 7.4.1. Given an optimization problem p-Π we define the (partial) function $\mathbf{p}_{p\text{-}\Pi}$ that given a graph G, outputs

$$\mathbf{p}_{p\text{-}\Pi}(G) = \bullet\{k \mid (G,k) \in p\text{-}\Pi\}$$

where $\bullet$ is interpreted as min or max depending whether p-Π is a minimization or a maximization problem.

Notice that $\mathbf{p}_{p\text{-}\Pi}$ is not defined when $\{k \mid (G,k) \in p\text{-}\Pi\}$ is an empty set.

Definition 7.4.2. We define $\mathsf{sol}_{p\text{-}\Pi}$ to be a (partial) function that, given as an input a graph G, returns a set S of size $\mathbf{p}_{p\text{-}\Pi}(G)$ such that $f(G,S) =$ **true**, and is not defined if no such set S exists.

The functions $\mathbf{p}_\Pi$ and sol_Π have the same domain.

Definition 7.4.3. Let p-Π be an optimization probem. We say that p-Π is *minor-closed* if $\mathbf{p}_{p\text{-}\Pi}$ is minor-closed. Moreover, we say that p-Π is *minor-bidimensional* if the following conditions hold:

1. p-Π is minor closed.
2. $\exists \delta \in \mathbb{R}^+ \, \exists k_0 \in \mathbb{N} \, \forall k \geq k_0, \frac{\mathbf{p}_{p\text{-}\Pi}(\boxplus_k)}{k^2} \geq \delta$.

Definition 7.4.4. Let $\mathbf{p}$ be a parameter. We say that $\mathbf{p}$ is *treewidth modulable* if there are constants c_1 and c_2 such that for every graph $G \in \mathsf{dom}(\mathbf{p})$ there is a $S \subseteq V(G)$ such that $|S| \leq c_1 \cdot \mathbf{p}(G)$ and $\mathbf{tw}(G \setminus S) \leq c_2$.

Definition 7.4.5. Let p-Π be an optimization problem. We say that p-Π is *treewidth modulable* if $\mathbf{p}_{p\text{-}\Pi}$ is treewidth modulable.

Definition 7.4.6. Let $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ be a function. We say that an optimization problem p-Π is *f-separable* if for any graph G and subset $L \subseteq V(G)$ such that $|\partial_G(L)| \leq t$, it holds that

$$|\mathsf{sol}_{p\text{-}\Pi}(G) \cap L| - f(t) \leq \mathbf{p}_{p\text{-}\Pi}(G[L]) \leq |\mathsf{sol}_{p\text{-}\Pi}(G) \cap L| + f(t).$$

p-Π is called *separable* if there exists a function f such that p-Π is *f-separable*. Moreover, p-Π is called *linearly separable* if there exists a constant c such that p-Π is $c \cdot t$-separable.

Definition 7.4.7. We say that an optimization problem p-Π is *protrusion decomposable* if $\mathbf{p}_{p\text{-}\Pi}$ is protrusion decomposable.

Definition 7.4.8. We say that an optimization problem p-Π has FII if $\mathbf{p}_{p\text{-}\Pi}$ has FII.

7.4.2 Background: a master theorem for linear kernels

A "couple" of testbed problem

Let us make some remarks on kernelization. Unfortunately, not all problems that admit FPT-algorithms also admit polynomial –or, ideally, linear– kernels. In many cases, it is unavoidable to restrict the graph classes where problems are defined in order to permit the design of such kernels.

We will use as a testbed the family of $\mathcal{F}$-Covering problems (see Section 4.2.1), where $\mathcal{F}$ *contains connected graphs and at least one of them is planar*. Notice that this problems is a minimization graph problem. Indeed, the corresponding subset certifying function is f where $f(G, S)$ is true if and only if some graph in $\mathcal{F}$ is a minor of $G \setminus S$.

$\mathcal{F}$-Covering belongs in the more general class of p-**p**-Modification porblem, where **p** is a modification parameter (see Section 4.2). Also it can generate several known problems, depending on the choice of the set of $\mathcal{F}$. For instance, we can obtain p-Vertex Cover (for $\mathcal{F} = \{K_2\}$), p-Feedback Vertex Set (when $\mathcal{F} = \{K_3\}$), and p-Vertex Outerplanarization (when $\mathcal{F} = \{K_4, K_{2,3}\}$). In [171, 172], it was proven that $\mathcal{F}$-Covering admits a polynomial kernel of size $k^{c_{\mathcal{F}}}$ where $c_{\mathcal{F}}$ is a parameter depending on the defining class[2] $\mathcal{F}$. An interesting question that emerged is whether a kernel of size $c_{\mathcal{F}} \cdot k^{O(1)}$ exists, that is *uniformly polynomial* with respect to the contribution of $\mathcal{F}$. This question has been answered negatively in [177] (based on standard complexity assumptions) for certain, but not all, instantiations of $\mathcal{F}$.

[2]Here $\mathcal{F}$ can also contain graphs that are not connected.

Problem properties for linear kernels

The results in [177] indicate that, if we insist on the derivation of linear kernels for $\mathcal{F}$-Covering, we should restrict instances to certain graph classes. In this direction, a series of meta-algorithmic results have been developed in [155, 174–176, 180], for a wide family of graph problems, when restricted either to H-minor free graphs or to H-topological minor free graphs.

Notice that $\mathcal{F}$-Covering is minor-closed. It is also well known that $\mathcal{F}$-Covering is treewidth-modulable (e.g., [155, 180]).

The meta-algorothmic results in [155, 174, 175, 180] are based on the following *master theorem*.

Proposition 7.4.1 ([155]). *Let p-$\Pi \subseteq \mathcal{G} \times \mathbb{N}$ be an optimization problem. If p-Π is protrusion-decomposable and has* FII*, then p-Π admits a linear kernel.*

The property of having FII was defined in [152] (see also [149, 155, 156, 191]) and has been extensively used in several meta-algorirthmic results in parameterized complexity and kernelization [155, 174, 175, 180]. The family of graph problems that have FII is quite extended. Conditions that yield the FII property have been proposed in [155] and more recently in [175] where it was proven that FII can be implied by other (more easy to be checked) problem properties, such as the separability and the expressibility in CMSO. The proof of Proposition 7.4.1 is based on the fact that the FII property enables the existence of *protrusion replacers* that are algorithmic procedures permitting the replacement of parts of the input graph, of an instance (G, k), by smaller ones [155]. Each replacement is done in a way that the new instance created after this replacement is equivalent to the initial one. The protrusion decomposability guarantees that when these replacements are not any more possible, the equivalent instance consists of a graph of size $O(k)$ and a parameter $k' \leq k$.

It has been noticed that very few problems are protrusion decomposable in general, therefore Proposition 7.4.1 is typically applied to problems restricted to special classes of graphs. In [174, 175] a series of additional combinatorial requirements where given for p-Π in order to guarantee protrusion-decomposablility for H-minor free graphs (and thus make Proposition 7.4.1 applicable to more problems). These conditions were related to the separability property and to the *Bidimensionality Theory* (Proposition 7.4.2, see [174, 175, 182] for more details). Also, in [180] it was proven that when restricted to H-topological minor free graphs, treewidth-modulable problems became also protrusion-decomposable (Proposition 7.4.3). Therefore, if $\mathcal{C}$ is a class of H-topological minor free graphs, for some H, then the fact that p-Π is treewidth-modulable and has FII, implies that p-$\Pi \cap \mathcal{C}$ admits a linear kernel. This statement applies for a wide variety of problems. For instance, it is known that $\mathcal{F}$-Covering has FII (see [155, 180]). This fact, together with the treewidth-modulability of $\mathcal{F}$-Covering, implies that the restriction of $\mathcal{F}$-Covering to H-topological minor-free graphs admits a linear kernel.

Another example of the applicability of Proposition 7.4.1 is the following:

$\mathcal{F}$-Packing	
Input:	A graph G and a $k \in N$.
Parameter:	k.
Question:	Does G contain k disjoint graphs, each containing a graph in $\mathcal{F}$ as a minor?

Again, we consider this problem for sets $\mathcal{F}$ *that contain connected graphs where at least one of them is planar*. $\mathcal{F}$-Packing is a maximization graph problem: The subset certifying function is f, where $f(G, S)$ is true if and inly if G contains a collection of disjoint graphs, each containing some of the graphs in $\mathcal{F}$ as a minor and each containing at most one of the

vertices in S. In a sense, $\mathcal{F}$-Packing is dual to $\mathcal{F}$-Covering.

Similarly to the case of $\mathcal{F}$-Covering, this problem, restricted to H-minor-free graphs, has FII and is treewidth-modulable [155]. Therefore, using Proposition 7.4.1, this restriction admits a linear kernel [155, 174, 175].

We could add more examples of the applicability of Proposition 7.4.1 here, however we restrict our attention to $\mathcal{F}$-Covering and $\mathcal{F}$-Packing as they are already able to produce many known graph problems and, also, they are minor-closed, which is a requirement of Theorem 7.3.1.

Having presented the necessary background, and hinted what the motivation behind the results in this Chapter is, we can finally put Theorem 7.3.1 in the picture. Theorem 7.3.1 yields the following – more powerful – version of Proposition 7.4.1, for minor-closed graph problems.

Theorem 7.4.1. *Let p-Π be an optimization problem that is minor-closed, has* FII*, and is protrusion decomposable. Then p-Π admits a linear kernel that is parameter-invariant and minor-monotone.*

Notice that the only additional demand in order to make the kernels of Proposition 7.4.1 minor-monotone and parameter-invariant is minor-closedness. Interestingly, in the proof of Theroem 7.3.1, minor-closedness is necessary for proving, not only the minor-monotonicity (as someone would probably expect), but also the parameter-invariance.

The proof of Theorem 7.4.1, as well as some other consequences on kernels, is discussed in the following Section.

7.4.3 Consequences on kernels.

As mentioned above, Theorem 7.4.1 is an easy corollary of Theorem 7.3.1. Let us see why.

Proof of Theorem 7.4.1. The definition of optimization problems implies that $\mathbf{p}_{p\text{-}\Pi}$ is computable. Notice that the conditions of Theorem 7.3.1 are satisfied, therefore there exists an algorithm that given a n-vertex graph G, outputs – in polynomial on n number of steps – a graph G' where $\mathbf{p}_{p\text{-}\Pi}(G) = \mathbf{p}_{p\text{-}\Pi}(G')$, $G' \leq_{\mathrm{m}} G$, and $|G'| = O(\mathbf{p}_{p\text{-}\Pi}(G))$. This means that for every instance $(G,k) \in \mathcal{G} \times \mathbb{N}$, $(G,k) \in p\text{-}\Pi$ if and only if $(G',k) \in p\text{-}\Pi$, therefore this algorithm is a parameter-invariant kernelization. Moreover as $G' \leq_{\mathrm{m}} G$ it is also minor monotone. □

In the previous Section, we mentioned some of the following results.

Proposition 7.4.2 (Lemma 3.2 of [174] and Lemma 3.6 of [175])**.** *Let p-Π be an optimization problem and let $\mathcal{C}$ be a graph class that is H-minor-free for some graph H. If p-Π is*

1. *minor-bidimensional,*
2. *CMSO-definable, and*
3. *linearly separable,*

then p-$\Pi \Cap \mathcal{C}$ is treewidth modulable.

Proposition 7.4.3 ([180])**.** *Let p-Π be an optimization problem and let $\mathcal{C}$ be a graph class that is H-topological minor-free for some graph H. If p-Π is treewidth modulable then p-$\Pi \Cap \mathcal{C}$ is protrusion decomposable.*

Proposition 7.4.4 (Theorem 4.4 of [175])**.** *If an optimization problem p-Π is CMSO-definable and separable, then it has* FII.

We are now able to investigate the prerequisites for the existence of minor-monotone and parameter invariant kernels. Theorem 7.4.1 has the following corollaries.

Corollary 7.4.1. Let p-Π be an optimization problem and let $\mathcal{C}$ be a graph class that is H-minor-free for some graph H. If p-Π is

1. minor-bidimensional,
2. CMSO-definable, and
3. linearly separable,

then p-$\Pi \Cap \mathcal{C}$ admits a linear, minor-monotone, and parameter-invariant kernel.

Proof. Let p-$\Pi' = p$-$\Pi \Cap \mathcal{C}$. The fact that $\mathcal{C}$ is CMSO-definable and Condition 2, implies that Condition 2 holds for p-Π' as well. Notice also that conditions 1 and 3 hold for p-Π' as well. CMSO-definability and separability of p-Π' together with Proposition 7.4.4 imply that p-Π' has FII. As p-Π' is minor-bidimensional, p-Π' is also minor-closed. Moreover, the minor-bidimensionality of p-Π' and the linear-separability of p-Π' imply that p-Π' is treewidth modulable, because of Proposition 7.4.2. From Proposition 7.4.3 on p-Π' we derive that p-Π' is protrusion decomposable. Finally, the result follows from Theorem 7.4.1. □

Using a subset of the arguments of the above proof we also derive the following – slightly more general – corollary.

Corollary 7.4.2. Let p-Π be an optimization problem and let $\mathcal{C}$ be a graph class that is H-topological minor-free for some graph H. If p-Π is

1. minor-closed,
2. treewidth modulable,
3. CMSO-definable, and
4. separable,

then p-$\Pi \Cap \mathcal{C}$ admits a linear, minor-monotone, and parameter-invariant kernel.

7.4.4 Consequences on obstructions

c-normal parameters

In order to exploit Theorem 7.4.1 for computing obstruction sets, it is necessary that the parameters we use Theorem 7.4.1 on, behave in a "normal" way. "Normal" in this set-up is defined as follows.

Definition 7.4.9. Let **p** be a graph parameter and $c \geq 1$ a constant. We say that **p** is *c-normal for*

- *vertex deletions* if for every graph $G \in \mathsf{dom}(\mathbf{p})$ and every $x \in V(G)$, $\mathbf{p}(G \setminus x) \geq p(G) - c$,
- *edge deletions* if for every graph $G \in \mathsf{dom}(\mathbf{p})$ and every $e \in E(G)$, $\mathbf{p}(G \setminus e) \geq \mathbf{p}(G) - c$, and
- *edge contractions* if for every graph $G \in \mathsf{dom}(\mathbf{p})$ and every $e \in E(G)$, $\mathbf{p}(G/e) \geq \mathbf{p}(G) - c$.

Furthermore, we say that **p** is *normal* if there exists a constant $c \geq 1$ such that **p** is c-normal for some of the above operations[3].

All parameters defined so far are (trivially) normal. Let us see an example.

Example 7.4.1. Treewidth and pathwidth are 1-normal for vertex deletions. We will prove the treewidth case. Let x be a vertex of a graph G, and assume that $\mathbf{tw}(G \setminus x) < \mathbf{tw}(G) - 1$. This means that there exist a tree-decomposition (T, B) of $G \setminus x$ with $\mathsf{width}(T, B) \leq \mathbf{tw}(G) - 2$. Notice that the tree-decomposition (T, B'), where, for every $t \in V(T)$, $B'_t = B \cup \{x\}$, is a tree-decomposition of G with $\mathsf{width}(T, B') \leq \mathbf{tw}(G) - 1$, a contradiction.

[3] We may also say that **p** is c-normal, without explicitly stating the local transformation, as – in all known cases – it does not matter what this transformation is.

In this Section we will prove that a minor-monotone and parameter-invariant kernel for a minor-closed problem p-Π, can make the computation of the set $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathbf{p}_{p\text{-}\Pi}, k)$ possible, as long as $\mathbf{p}_{p\text{-}\Pi}$ is a normal parameter. The rationale supporting this stems from the following lemma.

Lemma 7.4.1. Let p-Π be an minor-closed optimization problem that admits a kernel of size g that is minor-monotone and parameter-invariant. For every graph G there is a graph G' such that $G' \leq_{\mathrm{m}} G$, $|G'| \leq g(\mathbf{p}_{p\text{-}\Pi}(G))$, and $\mathbf{p}_{p\text{-}\Pi}(G') = \mathbf{p}_{p\text{-}\Pi}(G)$.

Proof. Let G be a graph and let $\mathbf{p}_{p\text{-}\Pi}(G) = \ell$. Notice that $(G, \ell) \in p$-Π. We distinguish two cases.

Case 1: p-Π is a minimization problem. We observe that $(G, \ell-1) \notin p$-Π. As p-Π admits a minor-monotone and parameter-invariant kernel of size g, if we run the kernelization algorithm with input $(G, \ell - 1)$ the output is a pair (G', ℓ') where $\ell' = \ell - 1$, $G' \leq_{\mathrm{m}} G$, and $|G'| \leq g(\ell - 1)$. As $(G, \ell - 1) \notin p$-Π we also have that $(G', \ell - 1) \notin p$-Π. The latter implies that $\mathbf{p}_{p\text{-}\Pi}(G') \geq \ell$ because p-Π is a minimization problem. Moreover, as $\mathbf{p}_{p\text{-}\Pi}$ is minor-closed, then $\mathbf{p}_{p\text{-}\Pi}(G') \leq \ell$, therefore $\mathbf{p}_{p\text{-}\Pi}(G') = \ell$.

Case 2: p-Π is a maximization problem. We observe that $(G, \ell) \in p$-Π. As p-Π admits a minor-monotone and parameter-invariant kernel of size g, if we run the kernelization algorithm with input (G, ℓ) the output is a pair (G', ℓ') where $\ell' = \ell$, $G' \leq_{\mathrm{m}} G$, and $|G'| \leq g(\ell)$. As $(G, \ell) \in p$-Π we also have that $(G', \ell) \in p$-Π. The latter implies that $\mathbf{p}_{p\text{-}\Pi}(G') \geq \ell$, because p-Π is a maximization problem. Moreover, as $\mathbf{p}_{p\text{-}\Pi}$ is minor-closed, then $\mathbf{p}_{p\text{-}\Pi}(G') \leq \ell$, therefore $\mathbf{p}_{p\text{-}\Pi}(G') = \ell$. □

Theorem 7.4.2. *Let p-Π be an minor-closed optimization problem that admits a kernel of size g that is minor-monotone and parameter-invariant. If $\mathbf{p}_{p\text{-}\Pi}$ is c-normal then $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathbf{p}_{p\text{-}\Pi}, k)$ is computable and, for every $k \in \mathbb{N}$, each graph in $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathbf{p}_{p\text{-}\Pi}, k)$ has at most $g(k - c)$ vertices*

Proof. We will prove that for every $G \in \mathsf{obs}_{\leq_\mathrm{m}}(\mathbf{p}_{p\text{-}\Pi}, k)$, $|G| \leq g(k-c)$. Since p-Π is an optimization problem it follows that $\mathbf{p}_{p\text{-}\Pi}$ is computable. Thus, we can check for every graph G of size at most $g(k-c)$ whether $\mathbf{p}_{p\text{-}\Pi}(G) > k$ and whether for every proper minor G' of G, $\mathbf{p}_{p\text{-}\Pi}(G') \leq k$, and, in this way, compute $G \in \mathsf{obs}_{\leq_\mathrm{m}}(\mathbf{p}_{p\text{-}\Pi}, k)$.

Let G' be a graph obtain from G according to Lemma 7.4.4. Since $G \in \mathsf{obs}_{\leq_\mathrm{m}}(\mathbf{p}_{p\text{-}\Pi}, k)$ and G' is a minor of G with $\mathbf{p}_{p\text{-}\Pi}(G') = \mathbf{p}_{p\text{-}\Pi}(G) > k$, it follows that $G = G'$. Therefore $|G| \leq g(\mathbf{p}_{p\text{-}\Pi}(G))$.

We assume that $\mathbf{p}_{p\text{-}\Pi}$ is c-normal for vertex deletions (the same argument holds for edge deletions or edge contractions), and let $u \in V(G)$. Then, it holds that

$$\mathbf{p}_{p\text{-}\Pi}(G) - c \leq \mathbf{p}_{p\text{-}\Pi}(G \setminus u)$$

and, as G is a minor minimal graph with $\mathbf{p}_{p\text{-}\Pi}$ greater than k and $G \setminus u \leq_\mathrm{m} G$, we conclude that

$$\mathbf{p}_{p\text{-}\Pi}(G \setminus u) \leq k.$$

This implies that $\mathbf{p}_{p\text{-}\Pi}(G) \leq k - c$. Hence $|G| \leq g(\mathbf{p}_{p\text{-}\Pi}(G)) \leq g(k - c)$. □

Theorem 7.4.1 can help us specify the properties an optimization problem p-Π must have to make the computation of $\mathsf{obs}_{\leq_\mathrm{m}}(\mathbf{p}_{p\text{-}\Pi}, k)$ possible.

Corollary 7.4.3. Let p-Π be an optimization problem that is minor-closed, has FII, and is protrusion decomposable. Then, if $p_{p\text{-}\Pi}$ is normal, then $\mathsf{obs}_{\leq_\mathrm{m}}(\mathbf{p}_{p\text{-}\Pi}, k)$ is computable and each graph in $\mathsf{obs}_{\leq_\mathrm{m}}(\mathbf{p}_{p\text{-}\Pi}, k)$ has at most $O(k)$ vertices.

This corollary also provides some knowledge on the graphs of the set $\mathsf{obs}_{\leq_\mathrm{m}}(\mathbf{p}_{p\text{-}\Pi}, k)$ when p-Π is the $\mathcal{F}$-Covering problem. As it is proved in [171, 172], each graph in $\mathsf{obs}_{\leq_\mathrm{m}}(\mathbf{p}_{p\text{-}\Pi}, k)$ has at most $k^{c_\mathcal{F}}$ vertices where $c_\mathcal{F}$ is a constant depending only on the class $\mathcal{F}$. An interesting open problem – that is somehow parallel to the problem of the size of the best kernel

for p-Π (examined in [177]) – is whether this upper bound can be reduced to $c_{\mathcal{F}} \cdot k^{O(1)}$. Theorem 7.4.1, combined with the results in [180], implies that for every H, each of the H-topological minor-free obstructions in $\mathsf{obs}_{\leq_m}(\mathbf{p}_{p\text{-}\Pi}, k)$ has at most $c_{\mathcal{F}} \cdot k$ vertices, for some computable constant $c_{\mathcal{F}}$. Similar conclusions can be derived when p-Π is the $\mathcal{F}$-Packing problem, but now for H-minor free graphs (using results in [44, 155]).

Distance to p parameters

Recall the modification parameter $(\mathbf{p}, r)$-dist, which is defined using the "host" parameter **p** (see Definition 4.2.4). We will implement Theorem 7.4.1 to compute the subset of $\mathsf{obs}_{\leq_m}((\mathbf{p}, r)\text{-dist}, k)$ containing H-topological minors-free graphs, for some graph H.

Definition 7.4.10. Let **p** be a parameter, and let $\mathcal{C}_G$ be the set containing the connected-components of a graph G.

- **p** is a *max-parameter* if $\mathbf{p}(G) = \max\{\mathbf{p}(C) \mid C \in \mathcal{C}_G\}$.
- **p** is a *sum-parameter* if $\mathbf{p}(G) = \sum_{C \in \mathcal{C}_G} \mathbf{p}(C)$.

Example 7.4.2. A characteristic max-parameter is **tw** and a characteristic sum-patrameter is **vc**.

We have to mention that – typically – layout parameters are max-parameters and vertex removal parameters are sum-parameters.

Definition 7.4.11. A set $\mathcal{F}$ of graphs is *connected* if and only if all graphs in $\mathcal{F}$ are connected.

Proposition 7.4.5 (Lemma 8.4 in [155])**.** *Let $\mathcal{F}$ be a connected set of graphs, that contains at least one planar graph. Then the $\mathcal{F}$-Covering problem has* FII.

We now present two theorems similar to Theorem 4.2.1.

Theorem 7.4.3. *Let $r, k \in \mathbb{N}$ and let $\mathbf{p}$ be a minor closed, computable, max-parameter that is big in grids. And let $\mathcal{C}$ be a graph class that is H-topological minor-free, for some graph H. Then the set*

$$\mathsf{obs}_{\leq_{\mathrm{m}}}((\mathbf{p}, r)\text{-}\mathsf{dist}, k) \cap \mathcal{C}$$

can be computed.

Proof. Let $r \in \mathbb{N}$ and let p-Π be the $(\mathbf{p}, r)$-Distance problem. First we prove that p-$\Pi \Cap \mathcal{C}$ satisfies the properties of Theorem 7.4.1.

p-Π is trivially minor closed. Notice that, from Observation 4.2.2, p-Π is the $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathbf{p}, r)$-Covering problem. Therefore, it suffices to show that $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathbf{p}, r)$-Covering has FII and for this, according to Proposition 7.4.5, it suffices to show that $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathbf{p}, r)$ is connected and contains a planar graph. $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathbf{p}, r)$ is connected because $\mathbf{p}$ is a max-parameter, therefore every minor-minimal graph G with $\mathbf{p}(G) > r$ must be connected. Since $\mathbf{p}$ is also big in grids, from Lemma 4.2.1, it follows that $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathbf{p}, r)$ contains a planar graph.

We will now prove that p-Π is treewidth modulable. Given a graph G, from the definitions of $(\mathbf{p}, r)$-dist and $\mathbf{p}_{p\text{-}\Pi}$, we know that there exist a set $S \subseteq V(G)$, of size at most $\mathbf{p}_{p\text{-}\Pi}(G)$, such that $\mathbf{p}(G \setminus S) \leq r$. From Lemma 4.2.1 it holds that $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathbf{p}, r)$ contains a planar graph, say O. As $\mathbf{p}(G \setminus S) \leq r$, $G \setminus S$ does not contain O as a minor, therefore, from [69,92] there exist a constant N such that $\mathbf{tw}(G \setminus S) \leq N$.

Since p-$\Pi \Cap \mathcal{C}$ is also treewidth modulable, using Proposition 7.4.3, we can show that p-$\Pi \Cap \mathcal{C}$ is protrusion decomposable. Therefore, from Theorem 7.4.1 p-$\Pi \Cap \mathcal{C}$ admits a linear kernel that is parameter-invariant and minor-monotone.

Given a graph G and a vertex $x \in V(G)$, notice that $\mathbf{p}_{p\text{-}\Pi}(G \setminus x) = \mathbf{p}_{p\text{-}\Pi}(G) - 1$. Therefore, $\mathbf{p}_{p\text{-}\Pi}$ is 1-normal for vertex deletions. Therefore, we can implement Theorem 7.4.2 for p-$\Pi \Cap \mathcal{C}$ and conclude that $\mathsf{obs}_{\leq_{\mathrm{m}}}((\mathbf{p}, r)\text{-}\mathsf{dist}, k) \cap \mathcal{C}$ can be computed. □

7.4.5 Consequences for EPTAS

Parameterized complexity and *Approximation algorithms* have recently came together, producing very interesting results [158–160, 166, 173].

Definition 7.4.12 (Cesati and Trevisan, 1997 [159])**.** An *Efficient Polynomial Time Approximation Scheme* (EPTAS in short) for a parameterized graph problem p-Π is a collection of algorithms (varying for different values of ϵ) that, for every $\epsilon > 0$, return an $(1+\epsilon)$-approximation (or $(1-\epsilon)$-approximation, depending on whether p-Π is a minimization or a maximization problem) of $\mathbf{p}_{p\text{-}\Pi}(G)$ in $f(1/\epsilon) \cdot |G|^{O(1)}$ steps, where f is some computable function.

A direct consequence of Theorem 7.4.1 is the following.

Theorem 7.4.4. *Let p-$\Pi \subseteq \mathcal{G} \times \mathbb{N}$ be an optimization graph problem. If p-Π is minor-closed, protrusion-decomposable, and has* FII*, and an* EPTAS *running in $f(1/\epsilon)\cdot|G|^{O(1)}$ steps, then p-Π has also an* EPTAS *running in $|G|^{O(1)} + f(1/\epsilon) \cdot OPT^{O(1)}$ steps, where $OPT = \mathbf{p}_{p\text{-}\Pi}(G)$.*

Notice that for the derivation of the theorem above, the parameter-invariance property is only required. In [173], a powerful meta-algorithmic result yielded the existence of EPTAS for a wide family of minor-closed problems on H-minor free graphs. In fact, the results in [173] already proved that the corresponding graph problems are protrusion decomposable. This way, Theorem 7.4.4 provides some acceleration of the running times of the meta-algorithmic framework of [173], for minor-closed problems, including all problems that can be expressed by $\mathcal{F}$-Covering and $\mathcal{F}$-Packing.

7.5 Setting up the Proof

The main idea of the proof of Theorem 7.3.1 is to find parts of the graph that can be "compressed" without changing the value of **p**, and that are

large enough so that eventually the size of the graph will decrease significantly. To do this, we have to define new types of graph decompositions, as well as boundaried graph decompositions. Then, we have to distinguish pairs of vertices in these decompositions that make the aforementioned compression possible.

To set-up the proof (milestone number four) we need to introduce many different combinatorial notions.

7.5.1 Graph decompositions

Definition 7.5.1. Let (T, χ) be a tree-decomposition of a graph G. We say that (T, χ) is *lean* if for every $t \in \mathbb{N}$, every pair $u_1, u_2 \in V(T)$, and every $Z_i \subseteq \chi(u_i), i \in [2]$, where $|Z_1| = |Z_2|$, it holds that

- either there exists an edge $e = \{w_1, w_2\} \in E(iTj)$ such that $\chi(w_1) \cap \chi(w_2) < t$,
- or there exists a collection of t internally vertex-disjoint paths in G between Z_1 and Z_2.

Proposition 7.5.1 ([32, 60])**.** *Every graph G has a lean tree-decomposition of width* $\mathbf{tw}(G)$.

Definition 7.5.2. Let (T, χ) be a tree-decomposition of a graph G. We say that (T, χ) is *small* if $\forall \{i, j\} \in E(T)$, $\chi(i) \setminus \chi(j) \neq \emptyset$ and $\chi(j) \setminus \chi(i) \neq \emptyset$.

We need a series of lemmata on tree-decompositions. For the proof of the following, we copy [142, Lemma 11.9].

Lemma 7.5.1. There exists an algorithm that, given a graph G and a (lean) tree-decomposition (T, χ) of G of width at most $\mathbf{tw}(G)$, returns – in $O(|G|)$ steps – a small (and lean) tree-decomposition (T', χ') of G of width at most $\mathbf{tw}(G)$, where $|T'| \leq |G|$.

Proof. In order to make D small we apply the following transformation: If for some edge $e = \{a, b\}$ of T, it holds that $\chi(a) \subseteq \chi(b)$, then let $D = (T', \chi')$, where $T' = T \setminus e$ and $\chi' = \chi \setminus \{(a, \chi(a))\}$. It is easy to verify that if D is lean, this transformation creates again a lean tree-decomposition. Moreover, when this transformation cannot be applied any more, the resulting decomposition is small. The fact that a small tree-decomposition of G does not have more than $|G|$ nodes follows by induction on the number of vertices of G and using the fact that the bag of every leaf in a small tree-decomposition contains an element not contained in any other bag. □

Definition 7.5.3. An (α, β)-protrusion decomposition $\mathcal{P} = \{X_0, X_1, \ldots, X_\ell\}$ of a graph G is *tight* if for all $i \in [\ell]$, $\partial_G(R_i) = N_G(X_i)$.

Lemma 7.5.2. If a graph G has an (α, β)-protrusion decomposition $\mathcal{P} = \{X_0, X_1, \ldots, X_\ell\}$, then it also have a tight (α, β)-protrusion decomposition.

Proof. Let $\mathcal{P} = \{X_0, X_1, \ldots, X_\ell\}$ be an (α, β)-protrusion decomposition of G. Recall that $R_i = N_G[X_i], i \in [\ell]$. For every $i \in [\ell]$, we set $Z_i = N_G(X_i) \setminus \partial_G(R_i)$ and observe that the sets $Z_1, \ldots, Z_\ell$ are pairwise disjoint. Indeed, this follows from the fact that, for each $i \in [\ell]$, each vertex in Z_i is incident only with edges in $G[R_i]$. For the same reason, none of these vertices can be a vertex of some $\partial_G(R_j), j \in [\ell]$. We conclude that

$$\bigcup_{i \in [\ell]} (Z_i \cap \partial_G(R_i)) \quad = \quad \emptyset. \tag{7.3}$$

We define $X'_i = X_i \cup Z_i$, $i \in [\ell]$, and observe that

$$N_G[X'_i] \quad = \quad N_G[X_i] \text{ and} \tag{7.4}$$

$$N_G(X'_i) \quad = \quad \partial_G(R_i) \tag{7.5}$$

We also define

$$X_0' \;=\; X_0 \setminus \bigcup_{i\in[\ell]} Z_i. \tag{7.6}$$

We claim that $\mathcal{P}' = \{X_0', X_1', \ldots, X_\ell'\}$ is also an (α, β)-protrusion decomposition of G. As the sets in the collection $\{Z_1, \ldots, Z_\ell, X_1, \ldots, X_\ell\}$ are mutually disjoint we obtain that $\mathcal{P}'$ is a partition of $V(G)$. As $X_0' \subseteq X_0$, Condition 1 holds. (7.4) and the fact that $R_i = N_G(X_i)$ is a β-protrusion decomposition of G, imply Condition 2. To prove Condition 3, recall first that $\partial_G(R_i) \subseteq N_G(X_i) \subseteq X_0$. Combining this fact with (7.3) and (7.6), we obtain that $\partial_G(R_i) \subseteq X_0'$. From (7.5), we deduce that $N_G(X_i') \subseteq X_0'$ and, therefore, the claim holds.

From (7.4) and (7.5) we get the additional property that $\forall i \in [\ell]$, $N_G(X_i') = \partial_G(N_G[X_i'])$, therefore $\mathcal{P}'$ is tight. □

7.5.2 Rooted trees

We will work towards proving Theorem 7.3.1 on *rooted trees* defined from a graph decomposition. Let us set the necessary notation.

Definition 7.5.4. A *rooted tree* is a pair (T, r) where T is a tree and $r \in V(T)$

We denote by $\mathsf{Leaf}(T, r)$ the set of all leaves of T that are different than r.

Definition 7.5.5. Let (T, r) be a rooted tree. Given two vertices a, b of T, we write $a \leq_{T,r} b$ to denote that $a \in V(rTb)$ and, in this case, we say that b is a *descendant* of a in (T, r). We also write $a \neq_{T,r} b$ to denote that neither $a \leq_{T,r} b$ nor $b \leq_{T,r} a$ is true and, in this case, we say that a and b are *uncomparable* in (T, r). Moreover, we write $a <_{T,r} b$ to denote that $a \neq_{T,r} b$ and $a \leq_{T,r} b$.

Definition 7.5.6. Let (T, r) be a rooted tree. Given some $q \in V(T)$, we denote the *set of descendants* of q in (T, r) as $\mathsf{desc}_{T,r}(q)$. The *children* of a vertex $q \in T$, in (T, r) are the vertices in $\mathsf{desc}_{T,r}(q)$ that are adjacent to q and are denoted as $\mathsf{child}_{T,r}(q)$.

Definition 7.5.7. A rooted tree (T, r) is *binary* if every vertex of T has at most two children.

Definition 7.5.8. Let (T, r) be a rooted tree, and let $v \in V(T)$. The *depth* of v in (T, r) is $|V(rTv)|$ and is denoted by $\mathsf{depth}_{T,r}(v)$. Also the *height* of v in (T, r) is defined as $\mathsf{height}_{T,r}(v) = \max\{|V(vTl)| \mid l \in \mathsf{Leaf}(T, r)\}$.

7.5.3 Pair collections

The first type of *pairs* in a rooted tree we consider is – the rather simple – *vertical pairs*.

Definition 7.5.9. Let (T, r) be a rooted tree. We say that a pair (a, b) of $V(T) \times V(T)$ is a *vertical pair* of (T, r) if $a <_{T,r} b$ [4]. We call a (resp. b) the *upper* (resp. the *lower*) vertex of (a, b). If $\{a, b\}$ is an edge of T, then (a, b) is an *edge-pair*.

Definition 7.5.10. The *inner part* of a vertical pair (a, b) of (T, r) is

$$\mathsf{inner}_{T,r}(a, b) = \{b\} \cup (\mathsf{desc}_{T,r}(a)) \setminus \mathsf{desc}_{T,r}(b)$$

The *outer part* of (a, b) is defined as

$$\mathsf{outer}_{T,r}(a, b) = \{a, b\} \cup (V(T) \setminus \mathsf{inner}_{T,r}(a, b)).$$

The *capacity* of (a, b) is defined as $\mathsf{capacity}_{T,r}(a, b) = |\mathsf{inner}_{T,r}(a, b)|$.

[4] Notice that in a vertical pair (a, b), a and b should be different vertices.

Notice that in the tree $T[\mathsf{inner}_{T,r}(a,b)]$ the vertex b is a leave, and that $\mathsf{outer}_{T,r}(a,b) \cap \mathsf{inner}_{T,r}(a,b) = \{a,b\}$ (see Figure 7.2).

Definition 7.5.11. Two vertical pairs (a,b) and (a',b') of (T,r) are *non-interfering* if $a \neq_{T,r} a'$ or $b <_{T,r} a'$, or $b' <_{T,r} a$ (see Figure 7.2).

Definition 7.5.12. A *pair collection* of (T,r) is a set $\mathcal{C}$ of pairwise non-interfering vertical pairs of T. The *minimum* (resp. *maximum*) *capacity* of $\mathcal{C}$ is the minimum (resp. maximum) capacity of a vertical pair in $\mathcal{C}$.

Definition 7.5.13. If (a,b) is a vertical pair of (T,r), the (a,b)-*compression of* (T,r) is the rooted tree (T',r) where T' is obtained from T if we remove all vertices in $\mathsf{inner}_{T,r}(a,b) \setminus \{a,b\}$, identify a and b, and call this vertex a again. We denote this new rooted tree by $(T,r) \setminus (a,b)$.

Observation 7.5.1. If (T,r) is a binary rooted tree, then $(T,r) \setminus (a,b)$ is also a binary rooted tree.

Definition 7.5.14. Let (a,b) be a vertical pair of (T,r) and $x,y \in \mathsf{inner}_{T,r}(a,b)$ (x and y are not necessarily distinct). We say that x and y are (a,b)-*aligned* if either $V(xTy) \subseteq V(aTb)$ or $V(xTy) \cap V(aTb) \subseteq \{x,y\}$ (see Figure 7.2).

Notice that in every vertical pair (a,b), a and b are (a,b)-aligned.

The following lemma shows us the size of the capacity a pair (a,b) must have in order to be sure that it contains an (a,b)-aligned pair.

Lemma 7.5.3. Let (T,r) be a rooted binary tree and let (a,b) be a vertical pair of (T,r). For every $d \geq 2$, if (a,b) has capacity

$$(d-2) \cdot 2^{d-2} + 1,$$

then there are $x,y \in \mathsf{inner}_{T,r}(a,b)$ such that x and y are (a,b)-aligned and $|xTy| \geq d$.

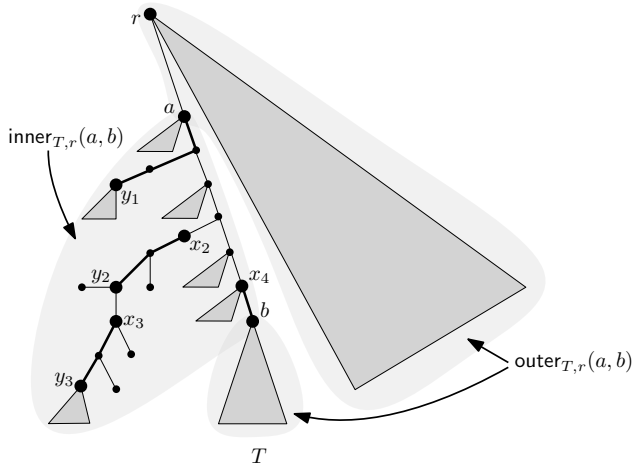


Figure 7.2: The vertical pairs $(x_2, y_2), (x_3, y_3)$ and (x_4, b) are non-inerfering. Notice that a and y_1 are not (a, b)-aligned.

Proof. We set $Y = T[\mathsf{inner}_{T,r}(a, b)]$. Observe that (Y, a) is a rooted graph where $b \in \mathsf{Leaf}(Y, a)$.

If $|aYb| \geq d$ then a and b are the required vertices of $\mathsf{inner}_{T,r}(a, b)$.

Suppose then that $|aYb| \leq d - 1$. Let $Y_1, \ldots, Y_z$ be the connected components of $Y \setminus E(aYb)$. Each Y_i is a binary tree whose root is some vertex of aYb that has only one child. Clearly, as $b \in \mathsf{Leaf}(Y, a)$ and $|aYb| \leq d - 1$, it holds that $z \leq d - 2$. Also the vertices of $Y_1, \ldots, Y_z$ form a partition of $V(Y)$. This means that one, say Y_i, of $Y_1, \ldots, Y_z$ has at least $1 + 2^{d-2}$ vertices. As Y_i is binary and its root has only one child, it follows that Y_i contains a path P on d vertices. Let x and y be its endpoints. We obtain that $|xYy| \geq d$. The lemma follows as the endpoints x and y are (a, b)-aligned. □

7.5.4 Boundaried graphs cont.

Recall the definitions of Section 7.2. Here, we focus on the relation $\equiv_{\mathbf{p},t}$ and present an algorithm which will later allow us to calculate the "cost"

of an (a, b)-compression (Lemma 7.5.4). By cost we mean the change in the value of the parameter $\mathbf{p}$. As we want to keep this value unchanged, we will have to look for pairs that have zero cost.

Definition 7.5.15. We define the *transposition function* $\mathsf{transp}_{\mathbf{p}}$ that receives as inputs pairs $(\mathbf{G}_1, \mathbf{G}_2)$ of boundaried graphs, where, $\mathbf{G}_1 \equiv_{\mathbf{p},t} \mathbf{G}_2$ and $\mathbf{G}_i \notin \mathsf{null}(\mathbf{p}), i \in [2]$, and outputs the constant $c_{\mathbf{G}_1,\mathbf{G}_2}$ in the equation (7.2).

Notice that, demanding $\mathbf{G}_i \notin \mathsf{null}(\mathbf{p}), i \in [2]$, guaranties that $\mathsf{transp}_{\mathbf{p}}$ $(\mathbf{G}_1, \mathbf{G}_2)$ is an integer.

Observation 7.5.2. Let $\mathbf{p}$ be a graph parameter and $t \in \mathbb{N}$. The following hold:

1. $\forall \mathbf{G}_1, \mathbf{G}_2 \in \mathcal{B}^{(t)}$, $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}_1, \mathbf{G}_2) = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_2, \mathbf{G}_1)$
2. $\forall \mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3 \in \mathcal{B}^{(t)}$, $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}_1, \mathbf{G}_3) = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_1, \mathbf{G}_2) + \mathsf{transp}_{\mathbf{p}}$ $(\mathbf{G}_2, \mathbf{G}_3)$

Lemma 7.5.4. Let $\mathbf{p}$ be a computable graph parameter that has FII. For every $t \in \mathbb{N}$, there exists an algorithm that, given a $\mathbf{G} \in \mathcal{B}^{(\leq t)}$, outputs $\mathbf{H} = \mathsf{rep}_{\mathbf{p}}(\mathbf{G})$ and $\mathsf{transp}_{\mathbf{p}}(\mathbf{H}, \mathbf{G})$.

Proof. Let $\mathcal{R}$ be a t-representative collection for $\equiv_{\mathbf{p},t}$.We prove first that there is an algorithm that, with input $\mathbf{H}, \mathbf{G} \in \mathcal{B}^{(\leq t)}$, checks whether $\mathbf{H} \equiv_{\mathbf{p},t}$ $\mathbf{G}$ in $O_t(1)$ steps. For this, we claim first that (7.2) is equivalent to the following:

$$\exists c_{\mathbf{G},\mathbf{H}} \in \mathbb{Z} \ \forall \mathbf{B} \in \mathcal{R} \cap \mathcal{B}^{(t')}, \quad \mathbf{p}(\mathbf{G} \oplus \mathbf{B}) - \mathbf{p}(\mathbf{H} \oplus \mathbf{B}) = c_{\mathbf{G},\mathbf{H}} \qquad (7.7)$$

Given that (7.2) $\Leftrightarrow$ (7.7), we check $\mathbf{H} \equiv_{\mathbf{p},t} \mathbf{G}$ as follows: Check first whether the underlying graphs of $\mathbf{H}$ and $\mathbf{G}$ both belong in the domain

of $\mathbf{p}$ (this is possible as $\mathbf{p}$ is computable), second, check whether $\mathbf{H}$ are compatible t'-boundaried graphs, for some $t' \in \{0\} \cup [t]$, and, finally, check whether (7.7) is correct (instead of (7.2)). Notice that (7.7) can be checked in $O_{\mathsf{card}_{\mathbf{p}}(t),|\mathbf{H}|,|\mathbf{G}|}(1)$ steps.

The "(7.2) $\Rightarrow$ (7.7)" direction is a direct consequence of the definition of $\equiv_{\mathbf{p},t}$ and the fact that $\mathcal{R} \cap \mathbf{B}^{(t)} \subseteq \mathbf{B}^{(t)}$. To prove the "(7.2) $\Leftarrow$ (7.7)" direction, observe first that for every $\mathbf{F} \in \mathcal{B}^{(t')}$, there exists some $\mathbf{B}_{\mathbf{F}} \in \mathcal{R} \cap \mathcal{B}^{(t')}$ such that $\mathbf{F} \equiv_{\mathbf{p},t} \mathbf{B}_{\mathbf{F}}$. This, together with the definition of $\equiv_{\mathbf{p},t}$ means that:

$$\begin{aligned}\forall \mathbf{F} \in \mathcal{B}^{(t')}\ \exists \mathbf{B}_{\mathbf{F}} \in \mathcal{R} \cap \mathcal{B}^{(t')}\ \exists c_{\mathbf{F},\mathbf{B}_{\mathbf{F}}} \in \mathbb{Z}\ \forall \mathbf{B} \in \mathcal{B}^{(t')}: \\ \mathbf{p}(\mathbf{F} \oplus \mathbf{B}) - \mathbf{p}(\mathbf{B}_{\mathbf{F}} \oplus \mathbf{B}) = c_{\mathbf{F},\mathbf{B}_{\mathbf{F}}}\end{aligned} \tag{7.8}$$

By applying (7.8) for $\mathbf{B} = \mathbf{G}$ and $\mathbf{B} = \mathbf{H}$, we have that:

$$\begin{aligned}\forall \mathbf{F} \in \mathcal{B}^{(t')}\ \exists \mathbf{B}_{\mathbf{F}} \in \mathcal{R} \cap \mathcal{B}^{(t')}\ \exists c_{\mathbf{F},\mathbf{B}_{\mathbf{F}}} \in\in \mathbb{Z}: \\ \mathbf{p}(\mathbf{F} \oplus \mathbf{G}) - \mathbf{p}(\mathbf{B}_{\mathbf{F}} \oplus \mathbf{G}) = c_{\mathbf{F},\mathbf{B}_{\mathbf{F}}}\end{aligned} \tag{7.9}$$

$$\begin{aligned}\forall \mathbf{F} \in \mathcal{B}^{(t')}\ \exists \mathbf{B}_{\mathbf{F}} \in \mathcal{R} \cap \mathcal{B}^{(t')}\ \exists c_{\mathbf{F},\mathbf{B}_{\mathbf{F}}} \in \mathbb{Z}: \\ \mathbf{p}(\mathbf{F} \oplus \mathbf{H}) - \mathbf{p}(\mathbf{B}_{\mathbf{F}} \oplus \mathbf{H}) = c_{\mathbf{F},\mathbf{B}_{\mathbf{F}}}\end{aligned} \tag{7.10}$$

From (7.9) and (7.10), we deduce the folowing:

$$\begin{aligned}\forall \mathbf{F} \in \mathcal{B}^{(t')}\ \exists \mathbf{B}_{\mathbf{F}} \in \mathcal{R} \cap \mathcal{B}^{(t')}: \\ \mathbf{p}(\mathbf{F} \oplus \mathbf{G}) - \mathbf{p}(\mathbf{B}_{\mathbf{F}} \oplus \mathbf{G}) = \mathbf{p}(\mathbf{F} \oplus \mathbf{H}) \\ - \mathbf{p}(\mathbf{B}_{\mathbf{F}} \oplus \mathbf{H})\end{aligned} \tag{7.11}$$

We now apply (7.7) for $\mathbf{B} = \mathbf{B}_{\mathbf{F}}$:

$$\exists c_{\mathbf{G},\mathbf{H}} \in \mathbb{Z}, \quad \mathbf{p}(\mathbf{G} \oplus \mathbf{B}_{\mathbf{F}}) - \mathbf{p}(\mathbf{H} \oplus \mathbf{B}_{\mathbf{F}}) = c_{\mathbf{G},\mathbf{H}} \tag{7.12}$$

Combining (7.11) and (7.12) we have:

$$\exists c_{\mathbf{G},\mathbf{H}} \in \mathbb{Z} \;\; \forall \mathbf{F} \in \mathcal{B}^{(t')} \;\; \exists \mathbf{B}_{\mathbf{F}} \in \mathcal{R}, \\ \mathbf{p}(\mathbf{F} \oplus \mathbf{G}) - \mathbf{p}(\mathbf{F} \oplus \mathbf{H}) = c_{\mathbf{G},\mathbf{H}} \qquad (7.13)$$

By simplyfying (7.13), we obtain:

$$\exists c_{\mathbf{G},\mathbf{H}} \in \mathbb{Z} \;\; \forall \mathbf{F} \in \mathcal{B}^{(t')}, \;\; \mathbf{p}(\mathbf{F} \oplus \mathbf{G}) - \mathbf{p}(\mathbf{F} \oplus \mathbf{H}) = c_{\mathbf{G},\mathbf{H}} \qquad (7.14)$$

which is equivalent to (7.2).

Observe now that as $\mathbf{p}$ is computable, there exists an algorithm that checks whether (7.7) is correct for some $\mathbf{H} \in \mathcal{R}$ that is compatible with $\mathbf{G}$. The algorithm then returns such a boundaried graph $\mathbf{H}$, along with the value $\mathsf{transp}_{\mathbf{p}}(\mathbf{H}, \mathbf{G}) = c_{\mathbf{G},\mathbf{H}}$. □

This lemma is the reason why Theorem 7.3.1 is not constructive. In the first line of the proof we assume that $\mathcal{R}$ is given to us. Unfortunately, there is no known general way to compute a t-representative collection given only the parameter $\mathbf{p}$ and a $t \in \mathbb{N}$. However, if $\mathsf{card}_{\mathbf{p}}$ is a computable function we can compute for every $t \in \mathbb{N}$ a t-representative collection, as Lemma 7.7.1 shows us.

We close this Section with a couple more definitions.

Definition 7.5.16. Let $\mathbf{G} = (G, X, \lambda)$ be a boundaried graph and $S \subseteq V(G)$. We define the boundaried graph $\mathbf{G}' = \mathbf{G} \setminus S$ such that $\mathbf{G}' = (G', X', \lambda')$, $G' = G \setminus S$, $X' = X \setminus S$, and $\lambda' = \lambda|_{X'}$.

Definition 7.5.17. Let $\mathbf{G}_1 = (G_1, X_1, \lambda_1)$ and $\mathbf{G}_2 = (G_2, X_2, \lambda_2)$ be two t-boundaried graphs. We say that $\mathbf{G}_1$ is a *minor* of $\mathbf{G}_2$, denoted by $\mathbf{G}_1 \leq_{\mathrm{m}} \mathbf{G}_2$, if there is a function $\sigma : V(G_1) \to 2^{V(G_2)}$ where

(1) $\forall x, y \in V(G_1), \; x \neq y \Rightarrow \sigma(x) \cap \sigma(y) = \emptyset$,

(2) $\forall x \in V(G_1),\ G_2[\sigma(x)]$ is connected,

(3) $\forall \{x, y\} \in E(G_1),\ G_2[\sigma(x) \cup \sigma(y)]$ is connected, and

(4) $\forall i \in [t],\ \psi_{\mathbf{G}_2}^{-1}(i) \in \sigma(\psi_{\mathbf{G}_1}^{-1}(i))$.

Notice that if $\mathbf{G}_1 \leq_{\mathrm{m}} \mathbf{G}_2$, then $H_{\mathbf{G}_2}$ is a spanning subgraph of $H_{\mathbf{G}_1}$. Also notice that the graph H is a *minor* of G if $(H, \emptyset, \varnothing) \leq_{\mathrm{m}} (G, \emptyset, \varnothing)$.

7.5.5 The functions used in the proof

The constant $c_{\mathbf{p}}$ of Theorem 7.3.1 is defined from the following functions.

Definition 7.5.18. Given a graph parameter $\mathbf{p}$ that has FII, we define the functions $\tau_{\mathbf{p}}, \theta_{\mathbf{p}}, \mu_{\mathbf{p}}, \delta_{\mathbf{p}}, \xi_{\mathbf{p}} : \mathbb{N} \to \mathbb{N}$ such that:

$$\begin{aligned}
\tau_{\mathbf{p}}(x) &= \max\{\mathbf{p}(G) \mid G \text{ is a graph in } \mathsf{dom}(\mathbf{p}) \text{ where } |G| \leq x\},\\
\theta_{\mathbf{p}}(x) &= (\mathsf{card}_{\mathbf{p}}(x) \cdot x! + 1)^{x+1},\\
\mu_{\mathbf{p}}(x) &= (\theta_{\mathbf{p}}(x) - 2) \cdot 2^{\theta_{\mathbf{p}}(x)-2} + 1,\\
\delta_{\mathbf{p}}(x) &= x((4x \cdot \mu_{\mathbf{p}}(x) - 1)^2 + 4x \cdot \mu_{\mathbf{p}}(x)), \text{ and}\\
\xi_{\mathbf{p}}(x) &= \tau_{\mathbf{p}}(x \cdot (2^{4x \cdot \mu_{\mathbf{p}}(x)-1} - 1)).
\end{aligned}$$

Definition 7.5.19. Given a graph parameter $\mathbf{p}$ that has FII, we define the constant $c_{\mathbf{p}}$ to be

$$c_{\mathbf{p}} = (\delta_{\mathbf{p}}(2 \cdot \mathsf{dec}(\mathbf{p})) \cdot (2 \cdot \mathsf{dec}(\mathbf{p}) \cdot \xi_{\mathbf{p}}(2 \cdot \mathsf{dec}(\mathbf{p})) + 1) + \mathsf{dec}(\mathbf{p})).$$

Notice that the function $\mathsf{card}_{\mathbf{p}}$ is involved in the definition of $c_{\mathbf{p}}$. This is the reason why $c_{\mathbf{p}}$ cannot be constructively computed in Theorem 7.3.1.

Unfortunately, from now on, the reader will have to come back to this Section many time.

7.5.6 Tree-decompositions of boundaried graphs

In this Section we extend the definitions of treewidth and lean tree-decompositions to boundaried graphs. Also, we introduce a couple of new tree-decompositions for boundaried graphs, namely *binary* and *rooted* tree-decompositions. Then we prove some basic properties of these decompositions.

Definition 7.5.20. Let $\mathbf{G} = (G, X, \lambda)$ be a boundaried graph. A *tree-decomposition* of $\mathbf{G}$ is a triple $D = (T, \chi, r)$ where (T, χ) is a tree-decomposition of G and r is a vertex of T such that $\chi(r) = X$. The *width* of a tree-decomposition $D = (T, \chi, r)$ is the width of the tree-decomposition (T, χ). The treewidth of a boundaried graph $\mathbf{G}$ is the minimum width over all its tree-decompositions and is denoted by $\mathbf{tw}(\mathbf{G})$.

To make the proofs shorter we will use some "abbreviations". Let $D = (T, \chi, r)$ be a tree-decomposition of a boundaried graph $\mathbf{G}$:

- For each $q \in V(T)$, we set $t_q = |\chi(q)|$ and we denote the corresponding frontier graph by $H_q = H_{\mathbf{G}_q}$.
- We set $T_q = T[\mathsf{desc}_{T,r}(q)]$ and we denote by $\mathbf{G}_q$ the t_q-boundaried graph $(G_q, \chi(q), \lambda_q)$ where

$$G_q = G[\bigcup_{q' \in V(T_q)} \chi(q')] \quad \text{and} \quad \lambda_q = \lambda|_{V(G_q)}.$$

Notice that if $a, b \in V(T)$ and $a \leq_{T,r} b$, then G_b is a subgraph of G_a.

We also set:

- $V_q = V(G_q)$,
- $\tilde{V}_q = V_q \setminus \chi(q)$,
- $\overline{G}_q = G \setminus \tilde{V}_q$,

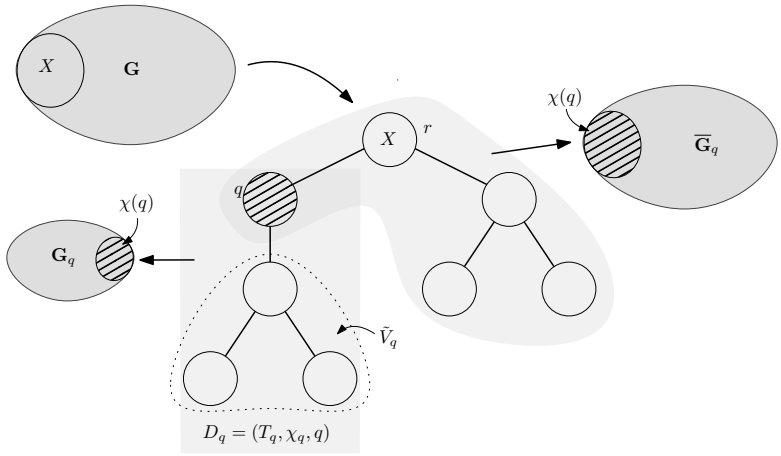


Figure 7.3: A boundaried graph $\mathbf{G} = (G, X, \lambda)$ and a tree-decomposition $D = (T, \chi, r)$ of it. Notice that $\tilde{V}_q$ contains the vertices of the three bags inside the dotted curve.

and we define the t_q-boundaried graph $\overline{\mathbf{G}}_q = (\overline{G}_q, \chi(q), \overline{\lambda}_q)$, where $\overline{\lambda}_q = \lambda|_{V(\overline{G}_q)}$, for $q \in V(T)$. Notice that $\mathbf{G}_q$ and $\overline{\mathbf{G}}_q$ are compatible and that $\mathbf{G}_q \oplus \overline{\mathbf{G}}_q = G$.

Finally, for every $q \in V(T)$, we use the notation $\overline{V}_q = V(\overline{G}_q)$, $\chi_q = \chi|_{V_q}$, and $D_q = (T_q, \chi_q, q)$ and observe that D_q is a tree-decomposition of $\mathbf{G}_q$ (see Figure 7.3).

Definition 7.5.21. We say that a tree-decomposition $D = (T, \chi, r)$ of a boundaried graph $\mathbf{G}$ is *lean* if (T, χ) is lean. $D = (T, \chi, r)$ is a *binary tree-decomposition* if (T, r) is a binary tree and for every edge-pair (i, j), if $|\chi(i)| = |\chi(j)|$, then G_j is a *proper* subgraph of G_i, i.e., $|G_j| < |G_i|$.

Lemma 7.5.5. Let $\mathbf{G}$ be a boundaried graph and let $D = (T, \chi, r)$ be a tree-decomposition of $\mathbf{G}$ of width $\leq t - 1$. Then $|\mathbf{G}| \leq t \cdot |T|$.

Proof. Notice that a set of $n = |\mathbf{G}|$ vertices is covered by $|T|$ sets, the bags of D, each of size at most t. As each bag of D covers at most t vertices, it follows that $n \leq t \cdot |T|$ as required. □

Lemma 7.5.6. Let $\mathbf{G}$ be a boundaried graph and let $D = (T, \chi, r)$ be a binary tree-decomposition of $\mathbf{G}$. Then $|T| \leq 4 \cdot |\mathbf{G}|$.

Proof. The proof is based on the fact that a binary tree-decomposition $D = (T, \chi, r)$ of $\mathbf{G}$ can be transformed to a decomposition $D' = (T', \chi', r)$ with the following properties:

A. T' is a binary tree.

B. If some vertex t of T has two children t_1 and t_2, then $\chi'(t) = \chi'(t_1) = \chi'(t_2)$.

C. If some vertex t of T has one child t', then either $|\chi'(t) \setminus \chi'(t')| = 1$ or $|\chi'(t') \setminus \chi'(t)| = 1$.

A tree-decomposition as above is called *nice* and it is known that $|T'| \leq 4 \cdot |\mathbf{G}|$ (e.g., [16]). It now remains to provide a way to transform (T, χ, r) to a nice tree-decomposition of $\mathbf{G}$ – of the same width – where $|T| \leq |T'|$. For this, we apply the following transformations as long as this is possible.

1. If $t \in V(T)$, t has two children and for one of them, say t', it holds that $\chi(t') \neq \chi(t)$, then subdivide the edge $\{t, t'\}$ and, if t_{new} is the subdivision vertex, set $\chi(v_{\text{new}}) = \chi(t)$.

2. If for some edge-pair (t, t') of T it holds that $|\chi(t) \cap \chi(t')| < \min\{|\chi(t)|, |\chi(t')|\}$, then subdivide the edge $\{t, t'\}$ and, if t_{new} is the subdivision vertex, set $\chi(v_{\text{new}}) = \chi(t) \cap \chi(t')$.

3. If for some edge $\{t, t'\}$ of T there are at least two vertices, a and a', in $\chi(t) \setminus \chi(t')$, then subdivide the edge $\{t, t'\}$ and, if t_{new} is the subdivision vertex, set $\chi(v_{\text{new}}) = \chi(t') \cup \{a\}$.

Let (T', χ', r) be the resulting tree-decomposition of $\mathbf{G}$. Clearly T' is binary, therefore property A holds. Property B follows by the fact that

Transformation 1 cannot be applied any more and Property C follows because transformations 2 and 3 cannot be applied any more. Therefore (T', χ', r) is a nice tree-decomposition, as required. □

LEMMA 7.5.7. There exists an algorithm that, given a boundaried graph $\mathbf{G}$ and a (lean) tree-decomposition $D = (T, \chi, r)$ of $\mathbf{G}$ with width at most $k \in \mathbb{N}$, outputs, in $O(|\mathbf{G}|)$ steps, a (lean) binary tree-decomposition of $\mathbf{G}$ – of width at most k – that has at most $2 \cdot |\mathbf{G}| - 1$ nodes.

Proof. Because of Lemma 7.5.1 we can assume that (T, χ) is small, and that T has at most $|\mathbf{G}|$ nodes. Clearly it holds that:

$$\forall \{x, y\} \in E(T), \;\; \chi(x) \setminus \chi(y) \neq \emptyset \text{ and } \chi(y) \setminus \chi(x) \neq \emptyset \qquad (7.15)$$

Let B be the set of vertices of T that have more than 2 children in T. For each $b \in B$ we apply the following transformation on D. Let $\{b_1, \ldots, b_s\}$ be the children of b in (T, r). We set $D' = (T', \chi', r)$ where:

- T' is obtained by T after removing the edges $\{b, b_i\}, i \in \{2, \ldots, s\}$, adding the new vertices $\{b'_2, \ldots, b'_{s-1}\}$, the edge $\{b, b'_2\}$, the edges $\{b'_i, b'_{i+1}\}, i \in \{2, \ldots, s-2\}$, the edges $\{b_i, b'_i\}, i \in \{2, \ldots, s-1\}$ and the edge $\{b'_{s-1}, b_s\}$ (see Figure 7.4), and
- $\chi' = \chi \cup \{(b'_i, \chi(b)) \mid i \in \{2, \ldots, s-1\}\}$.

We call $D' = (T', \chi', r)$ the tree-decomposition of $\mathbf{G}$ that is obtained after applying the above transformation for every $b \in B$. Notice that T is a binary tree and that the width of D' is equal to the width of D. Also, it is easy to observe that the number of new nodes is upper bounded by the number of leaves of T, therefore the number of nodes of the new decomposition cannot increase more than double. Moreover, it can be easily verified that if D is lean, then D' is lean as well.

Notice that D satisfied Condition (7.15) before the application of the above transformation. Moreover, for every vertical pair (a, b) where Condition (7.15) is still true after the transformation, it holds that $|G_b| < |G_a|$.

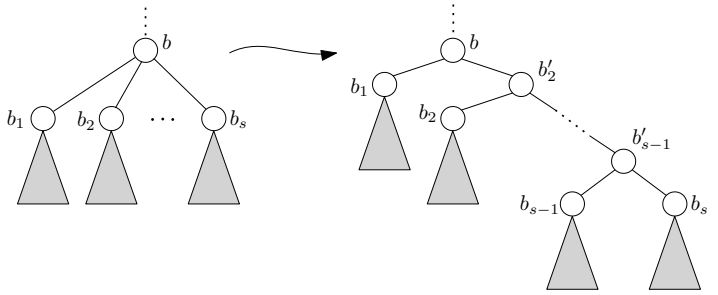


Figure 7.4: The transformation of Lemma 7.5.7.

If Condition (7.15) is not any more true for some vertical pair (x, y), created after the application of the above transformation, then $(x, y) \in \{(b'_i, b'_{i+1}) \mid i \in \{2, \ldots, s-2\}\}$ for some $b \in B$ (here we denote $b'_2 = b$). In this case, $\chi(b'_i) = \chi(b), i \in \{2, \ldots, s-1\}$. But then b'_i has another child, that is b_i, where both $\chi(b_i) \setminus \chi(b'_i)$ and $\chi(b'_i) \setminus \chi(b_i)$ are non-empty, because both $\chi(b_i) \setminus \chi(b)$ and $\chi(b) \setminus \chi(b_i)$ where initially non-empty. This implies that if $\chi(b'_i) = \chi(b'_{i+1})$, $V_{b'_{i+1}} \subsetneq V_{b'_i}$ then $|G_{b'_{i+1}}| < |G_{b'_i}|$ as required. Therefore D' is a binary tree-decomposition.

Notice now that the number of new nodes of D' is equal to

$$\sum_{v \in V_{\geq 3}} (\deg_T(v) - 2),$$

where $V_{\geq 3}$ is the number of vertices in T with more than 2 children. It is easy to observe that

$$\sum_{v \in V_{\geq 3}} (\deg_T(v) - 2) \leq \mathsf{Leaf}(T, r) \leq |T| - 1.$$

Therefore, $|T'| \leq 2 \cdot |T| - 1 \leq 2 \cdot |\mathbf{G}| - 1$. □

Lemma 7.5.8. Let $\mathbf{G} = (G, X, \lambda)$ be a boundaried graph where $G[X]$ is a clique and $\mathbf{tw}(G) \leq t - 1$. Then $\mathbf{G}$ has a lean binary tree-decomposition

$D = (T, \chi, r)$ – with width at most $t - 1$ – where $\chi(r) = X$ and $|T| \leq 2 \cdot |\mathbf{G}|$.

Proof. From Proposition 7.5.1, G has a lean tree-decomposition (T, χ) of width $\mathbf{tw}(G)$. As $G[X]$ is a clique, all vertices of X will go in the same "bag" of (T, χ). Let $q \in V(T)$ such that $X \subseteq \chi(q)$ and set $\mathbf{G}' = (G, \chi(q), \lambda)$. Notice that (T, χ, q) is a lean tree-decomposition of $\mathbf{G}'$ of width $\mathbf{tw}(G)$. From Lemma 7.5.7, (T, χ, q) can be transformed to a lean binary tree-decomposition $D = (T', \chi', q)$ of width $\mathbf{tw}(G)$, such that T' has at most $2 \cdot |\mathbf{G}'| - 1 = 2 \cdot |G| - 1$ nodes. Recall that $\chi'(q) = \chi(q)$.

In the case where $\chi(q) = X$, D is the required lean binary tree decomposition of $\mathbf{G}$ and we are done.

In the case where $X \subsetneq \chi(q)$, rename q to q_{old}, add in T a new vertex q_{new} along with the edge $\{q, q_{\text{new}}\}$, set $\chi = \chi \cup (q_{\text{new}}, X)$ and rename q_{new} to q. Notice that, as $X \subsetneq \chi(q)$, after this modification D remains a lean binary tree-decomposition of G of width $\mathbf{tw}(G)$. □

Lemma 7.5.9. There exists an algorithm that, given a boundaried graph $\mathbf{G} = (G, X, \lambda)$ and a tree-decomposition $D = (T', \chi')$ of $G' = G \setminus X$, of width $\mathbf{tw}(G')$, constructs, in $O(|\mathbf{G}|)$ steps a binary tree-decomposition of $\mathbf{G}$ – of width at most $\mathbf{tw}(G') + |X|$ – with at most $2 \cdot (|\mathbf{G}| - |X|) + \mathbf{tw}(G') + 1$ nodes.

Proof. Pick any vertex $\overline{r}$ of T' and let $S = \chi'(\overline{r})$. Clearly $D' = (T', \chi', \overline{r})$ is a rooted tree-decomposition of $\mathbf{G}' = (G', S, \lambda \setminus X)$. From Lemma 7.5.7, there is an algorithm that transforms D' to a binary tree-decomposition $D'' = (T'', \chi'', \overline{r})$ of $\mathbf{G}'$ of width $\mathbf{tw}(G')$ and $2 \cdot |\mathbf{G}'| = 2 \cdot (|\mathbf{G}| - |X|)$ nodes, in $O(|\mathbf{G}'|)$ steps. We update χ'' so that for every $q \in V(T'')$, $\chi''(q) = \chi''(q) \cup \{X\}$. Clearly, the new D'' is a binary rooted tree-decomposition of $\mathbf{G}$, where $\chi''(\overline{r}) = S \cup X$.

Let $s = |S|$ and pick a colection $A_1, \ldots, A_s$ of subsets of $S \cup X$ such that $X = A_s \subsetneq A_{s-1} \subsetneq \cdots \subsetneq A_1 \subsetneq S \cup X$. We use $D'' = (T'', \chi'', \overline{r})$ in

order to construct a binary tree-decomposition $D^+ = (T^+, \chi^+, r^+)$ of $\mathbf{G}$ of width $\mathbf{tw}(G')$ as follows.

Let T^+ be the disjoint union of T'' and a path on s vertices $q_1, \ldots, q_s$. The construction of T^+ is completed by adding the edge $\{\overline{r}, q_1\}$. We also set $r^+ = q_s$. Finally we set $\chi^+ = \chi'' \cup \{(q_i, A_i) \mid i \in [s]\}$ and notice that $D^+ = (T^+, \chi^+, r^+)$ is a binary tree-decomposition of $\mathbf{G}$ – of width at most $\mathbf{tw}(G') + |X|$ – and with at most $2 \cdot (|\mathbf{G}| - |X|) + \mathbf{tw}(G') + 1$ nodes, as $|S| \leq \mathbf{tw}(G') + 1$ □

Lemma 7.5.10. Let $\mathbf{G} = (G, X, \lambda) \in \mathcal{B}^{(\leq t)}$. There exists an algorithm that checks whether $\mathbf{tw}(\mathbf{G}) \leq t - 1$, and if so, outputs a binary tree-decomposition of $\mathbf{G}$ – of width at most $t - 1$ – in $O_t(|\mathbf{G}|)$ steps.

Proof. Let G' be the graph obtained from G after adding all edges missing from $G[X]$, i.e, $G'[X]$ is a clique of $|X|$ vertices. We use the algorithm of Theorem 4.1.2 to check whether G' has treewidth at most $t - 1$.

If the answer is affirmative, then this algorithm computes a tree-decomposition (T', χ') of G', with width at most $t - 1$, in linear time. As $G'[X]$ is a clique there exists a $r' \in V(T')$ such that $\chi'(r') \supseteq X$. Let r'' be a vertex not in $V(T')$. We define the tree $T'' = (V(T') \cup \{r''\}, E(T') \cup \{\{r'', r'\}\})$ and the tree-decomposition $D'' = (T'', \chi'', r'')$ of $\mathbf{G}$, where $\chi'' = \chi' \cup \{(r'', X)\}$. Notice that D'' was width at most $t - 1$.

On the other hand, if the answer of the algorithm is negative, that means there is no tree-decomposition of G' with width $t - 1$, and, therefore, there is no tree-decomposition of $\mathbf{G}$ with width at most $t - 1$ either.

Using D'' and the algorithm of Lemma 7.5.7 we can compute a binary tree-decomposition of $\mathbf{G}$, say $D = (T, \chi, r)$, with width at most $t - 1$, in $O_t(|\mathbf{G}|)$ steps. □

Definition 7.5.22. Let $\mathbf{G} = (G, X, \lambda)$ be a boundaried graph and let $D = (T, \chi, r)$ be a tree-decomposition of $\mathbf{G}$. Given $\alpha, \beta \in \mathbb{N}$, we say that D is an (α, β)-*tree-decomposition* of $\mathbf{G}$ if the following conditions hold (see Figure 7.5):

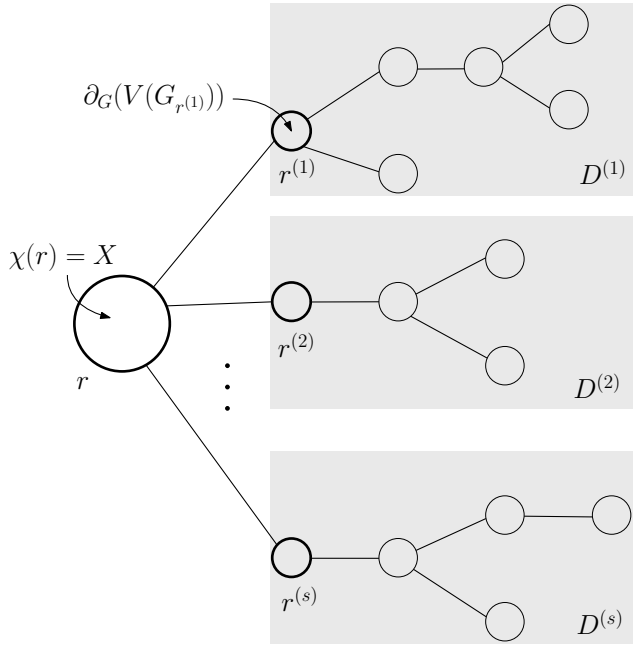


Figure 7.5: An (α, β)-tree-decomposition of a boundaried graph $\mathbf{G}$.

(1) $|\chi(r)| \leq \alpha$,

(2) if $\{r^{(1)}, \ldots, r^{(s)}\}$ is the set of children of r in (T, r), then $s \leq \alpha$,

(3) $\forall h \in [s]$, $D^{(h)} = (T_{r^{(h)}}, \chi_{r^{(h)}}, r^{(h)})$ is a binary tree-decom-position of $\mathbf{G}_{r^{(h)}}$ of width at most β,

(4) $\forall h \in [s]$, $\chi(r^{(h)}) = \partial_G(V(G_{r^{(h)}}))$, and

(5) for each $\{i, j\} \in \binom{[s]}{2}$, $\chi(r^{(i)}) \neq \chi(r^{(j)})$.

We also define t_q, H_q, T_q, $\mathbf{G}_q$, V_q, $\tilde{V}_q$, $\overline{\mathbf{G}}_q$ and D_q, for every $q \in V(T)$, as we did in the case of tree-decompositions.

(α, β)-tree-decompositions can be related to (α, β)-protrusion decompositions in the following way.

Lemma 7.5.11. Let G be a graph that has an (α, β)-protrusion decomposition. Then there exists some boundaried graph $\mathbf{G}$, whose underlying graph is G, such that $\mathbf{G}$ has a $(\alpha, 2\beta)$-tree-decomposition of at most $1 + 2 \cdot |\mathbf{G}| + \alpha \cdot \beta + \alpha$ nodes.

Proof. Let $\mathcal{P} = \{X_0, X_1, \ldots, X_\ell\}$ be an (α, β)-protrusion decomposition of G. From Lemma 7.5.2, we may assume that $\mathcal{P}$ is tight, i.e.,

$$\forall i \in [\ell], \quad N_G(X_i) = \partial_G(N_G[X_i]). \tag{7.16}$$

Recall that $R_i = N_G[X_i]$, $|\partial_G(R_i)| \leq \beta$ and $\mathbf{tw}(G[R_i]) \leq \beta$, $i \in [\ell]$. We set $G_i = G[X_i], i \in [\ell]$ and, as $X_i \subseteq R_i$, we obtain that $\mathbf{tw}(G_i) \leq \mathbf{tw}(G[R_i]), i \in [\ell]$. Notice that the vertex sets of the graphs in $G_1, \ldots, G_\ell$ are pairwise disjoint, therefore:

$$\forall I \subseteq [\ell], \quad \mathbf{tw}(\textstyle\bigcup_{i \in I} G_i) \leq \beta \tag{7.17}$$

We say that $G_i \sim G_j$ if and only if $\partial_G(R_i) = \partial_G(R_j)$. Clearly, $\sim$ defines an equivalence relation. Let $\{\mathcal{G}_1, \ldots, \mathcal{G}_s\}$ be the partition of $\{G_1, \ldots, G_\ell\}$ into the equivalence classes of $\sim$.

For $h \in [s]$, we set $G^{(h)} = \cup\, \mathcal{G}_h$ and observe that, because of 7.17, $\mathbf{tw}(G^{(h)}) \leq \beta$. Also, we set

$$\overline{G}^{(h)} = G[V(G^{(h)}) \cup Z^{(h)}],$$

where $Z^{(h)}$ is the common open neighbourhood, in G, of all the vertex sets of the graphs in $\mathcal{G}^{(h)}$, $h \in [s]$. From (7.16), $Z^{(h)} = \partial_G(V(\overline{G}^{(h)}))$. As each $Z^{(h)}$ is some of the sets in $\{\partial_G(R_1), \ldots, \partial_G(R_\ell)\}$, taking into account (7.16), we have that $|Z^{(h)}| \leq \beta, h \in [s]$.

Let λ be some labelling of G. We set, for every $h \in [s]$,

$$\mathbf{G}^{(h)} = (\overline{G}^{(h)}, Z^{(h)}, \lambda|_{V(\overline{G}^{(h)})})$$

and $t^{(h)} = |Z^{(h)}|$. Recall that $t^{(h)} \leq \beta$, $h \in [s]$. From Lemma 7.5.9, each $\mathbf{G}^{(h)}$ has a binary tree-decomposition $D^{(h)} = (T^{(h)}, \chi^{(h)}, r^{(h)})$ of width at most $\mathbf{tw}(G^{(h)}) + t^{(h)} \leq 2 \cdot \beta$ and at most

$$2 \cdot (|\mathbf{G}^{(h)}| - |Z^{(h)}|) + \mathbf{tw}(\mathbf{G}^{(h)}) + 1 \leq 2 \cdot (|\mathbf{G}^{(h)}| - |Z^{(h)}|) + \beta + 1$$

nodes.

We now construct a tree-decomposition $D = (T, \chi, r)$ of $\mathbf{G}$ as follows. To construct the tree T we take the disjoint union of $T^{(1)}, \ldots, T^{(s)}$, then we add a new vertex r and we make r adjacent with $r^{(1)}, \ldots, r^{(s)}$. We then define $\chi = \{(r, X_0)\} \cup \chi^{(1)} \cup \cdots \cup \chi^{(s)}$. Notice that D is a tree-decomposition of G. We claim that D is an $(\alpha, 2 \cdot \beta)$-decomposition.

Taking into account that $T^{(h)} = T_{r^{(h)}}$ and $\mathbf{G}^{(h)} = \mathbf{G}_{r^{(h)}}$, $h \in [\ell]$, it is easy to verify that D is an (α, β)-decomposition of G of at most

$$\begin{aligned} 1 + \sum_{h \in [s]} (2 \cdot (|\mathbf{G}^{(h)}| - |Z^{(h)}|) + \beta + 1) &= 1 + 2 \cdot \sum_{h \in [s]} (|\mathbf{G}^{(h)}| - |Z^{(h)}|) \\ &\quad + \sum_{h \in [s]} (\beta + 1) \\ &\leq 1 + 2 \cdot |\mathbf{G}| + \alpha \cdot (\beta + 1) \end{aligned}$$

nodes. Moreover, Condition 1 follows as $|\chi(r)| = |X_0| \leq \alpha$. Condition 2 follows from the fact that s is the number of equivalence classes of $\sim$, which is at most $\ell \leq \alpha$. Condition 3 follows directly by the construction of D. For Condition 4 it is enough to observe that for every $h \in [s]$, $\chi(r^{(h)}) = Z^{(h)} = \partial_G(V(\overline{G}^{(h)})) = \partial_G(V(G_{r^{(h)}}))$. As $\chi(r^{(h)}) = Z^{(h)}$, and, by their definition, all sets in $\{Z^{(1)}, \ldots, Z^{(s)}\}$ are pairwise distinct, Condition 5 is satisfied. □

7.5.7 A lemma on the compression of admissible pairs

We move on and define the type of pairs satisfying the properties that allow us to compress "parts" of a boundaried graph. But first we have to define what such a "part" is and then, how can we compress it.

Definition 7.5.23. Let $\mathbf{G}_i = (G_i, X_i, \lambda_i), i \in [2]$ be two boundaried graphs. We say that $\mathbf{G}_2$ is a *part of* $\mathbf{G}_1$ if G_2 is a subgraph of G_1 such that $V(G_2)$ is the union of some of the connected components of $G_1 \setminus X_2$ that do not contain vertices of X_1 and $\lambda_1|_{V(G_2)} = \lambda_2$.

Definition 7.5.24. Let $\mathbf{G}_2 = (G_2, X_2, \lambda_2)$ and $\mathbf{G}'_2 = (G'_2, X'_2, \lambda'_2)$ be boundaried graphs. We say that $\mathbf{G}'_2$ and $\mathbf{G}_2$ are *strongly compatible* if they are compatible, and $\lambda'_2(V(G_2)) \subseteq \lambda_2(V(G_2))$.

Definition 7.5.25. Let $\mathbf{G}_i = (G_i, X_i, \lambda_i), i \in [2]$ and $\mathbf{G}'_2 = (G'_2, X'_2, \lambda'_2)$ be boundaried graphs. If $\mathbf{G}_2$ is a part of $\mathbf{G}_1$ and $\mathbf{G}_2$ and $\mathbf{G}'_2$ are strongly compatible, we define the *replacement of* $\mathbf{G}_2$ *by* $\mathbf{G}'_2$ *in* $\mathbf{G}_1$ as the boundaried graph $\mathbf{G}'_1 = (G'_1, X'_1, \lambda'_1)$ where

- $G'_1 = (G_1 \setminus (V(G_2) \setminus X_2), X_2, \lambda \setminus (V(G_2) \setminus X_2)) \oplus \mathbf{G}'_2$,
- $X'_1 = X_1$, and
- $\lambda'_1 = (\lambda_1 \setminus (\lambda_2 \setminus (V(G_2) \setminus X_2))) \cup \lambda'_2$.

Definition 7.5.26. Let $\mathbf{G} = (G, X, \lambda)$ be a boundaried graph and let $D = (T, \chi, r)$ be a binary tree-decomposition of G of width at most $t-1$. We say that a vertical pair (a, b) of (T, r) is *compressible* for $(\mathbf{G}, D)$ if $\mathbf{G}_a$ and $\mathbf{G}_b$ are compatible.

Definition 7.5.27. Given a vertical pair (a, b) that is compressible for $(\mathbf{G}, D)$, we define the (a, b)*-compression of the pair* $(\mathbf{G}, D)$ as the pair $(\mathbf{G}', D')$ where $\mathbf{G}' = (G', X, \lambda')$ is the replacement of $\mathbf{G}_a$ by $\mathbf{G}_b$ in $\mathbf{G}$, $D = (T', \chi', r)$, $T' = T \setminus (a, b)$, and $\chi' = \chi|_{V(T')}$.

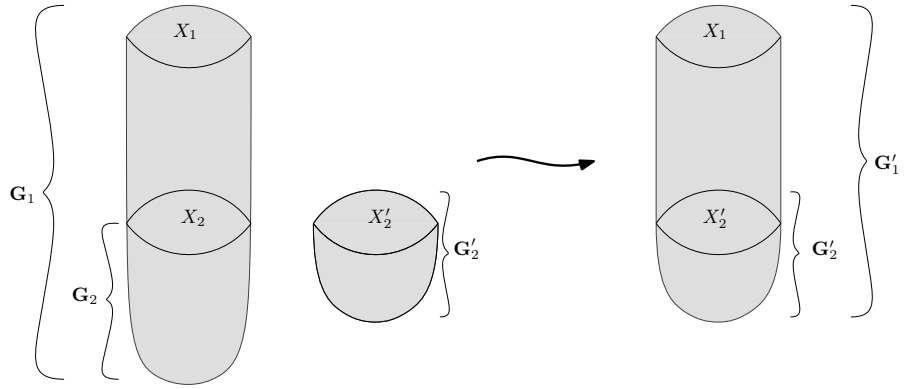


Figure 7.6: The replacement of $\mathbf{G}_2$ by $\mathbf{G}_2'$ in $\mathbf{G}_1$.

Notice that if D is a binary tree-decomposition of $\mathbf{G}$, then D' is again a binary tree-decomposition of $\mathbf{G}$.

Definition 7.5.28. Let $\mathbf{G} = (G, X, \lambda)$ be a boundaried graph and let $D = (T, \chi, r)$ be a binary tree-decomposition of G of width at most $t-1$. Let $t \in \mathbb{N}$ and $\mathbf{p}$ be a graph parameter that has FII. We say that a vertical pair (a, b) of (T, r) is $\equiv_{\mathbf{p},t}$-*admissible* for $(\mathbf{G}, D)$ if (a, b) is compressible for $(\mathbf{G}, D)$ and $\mathbf{G}_a \equiv_{\mathbf{p},t} \mathbf{G}_b$.

The following Lemma is crucial for the proofs in this Chapter, as it shows some useful properties of the replacement on admissible pairs.

Lemma 7.5.12. Let $\mathbf{p}$ be a graph parameter that has FII and let $t \in \mathbb{N}$. Let $\mathbf{G} = (G, X, \lambda)$ be a boundaried graph and let $D = (T, \chi, r)$ be a rooted tree-decomposition of $\mathbf{G}$ of width at most $t-1$. Let (i_1, i_2) be an $\equiv_{\mathbf{p},t}$-admissible pair for $(\mathbf{G}, D)$ where $\mathbf{G}_{i_1} \equiv_{\mathbf{p},t_{i_1}} \mathbf{G}_{i_2}$ and let $i_3 \in V(T)$ such that $i_2 \leq_{T,r} i_3$. Let also $\mathbf{G}_{\text{new}} = (G_{\text{new}}, X_{\text{new}}, \lambda_{\text{new}})$ be a boundaried graph that is strongly compatible with $\mathbf{G}_{i_3}$ and $\mathbf{G}_{i_3} \equiv_{\mathbf{p},t_{i_3}} \mathbf{G}_{\text{new}}$ and let $\mathbf{G}_{i_i}'$ be the replacement of $\mathbf{G}_{i_3}$ by $\mathbf{G}_{\text{new}}$ in $\mathbf{G}_{i_i}$, for $i \in [2]$. Then the following hold:

1. $\mathbf{G}_{i_1}' \equiv_{\mathbf{p},t_{i_1}} \mathbf{G}_{i_2}'$,

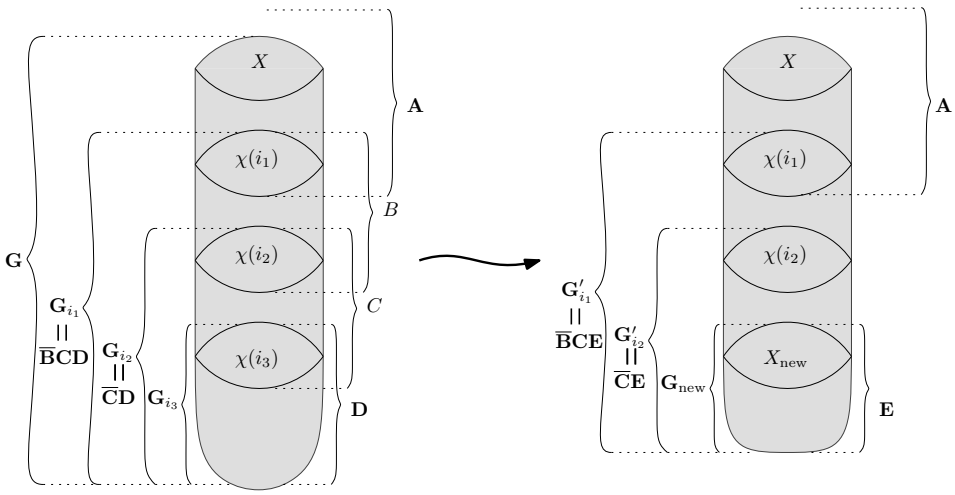


Figure 7.7: The boundaried graphs in Lemma 7.5.12.

2. $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}'_{i_1}, \mathbf{G}'_{i_2}) = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_{i_1}, \mathbf{G}_{i_2})$,

3. $\mathbf{G}_{i_2} \equiv_{\mathbf{p},t} \mathbf{G}'_{i_2}$, and

4. $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}_{i_2}, \mathbf{G}'_{i_2}) = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_{i_3}, \mathbf{G}_{\text{new}})$.

Proof. We define (see Figure 7.7):

$$\begin{aligned} B &= G_{i_1} \cap \overline{G}_{i_2} & C &= G_{i_2} \cap \overline{G}_{i_3} & D &= G_{i_3} \\ \overline{\mathbf{B}}\mathbf{C}\mathbf{E} &= \mathbf{G}'_{i_1} & \overline{\mathbf{C}}\mathbf{E} &= \mathbf{G}'_{i_2} & \overline{\mathbf{B}}\mathbf{C}\mathbf{D} &= \mathbf{G}_{i_1} \\ \overline{\mathbf{C}}\mathbf{D} &= \mathbf{G}_{i_2} & \mathbf{D} &= \mathbf{G}_{i_3} & \mathbf{E} &= \mathbf{G}_{\text{new}}. \end{aligned}$$

If $\mathbf{A} = (A, Y, \lambda)$ is some t_{i_1}-boundaried graph, we denote $\mathbf{A}\mathbf{B}\overline{\mathbf{C}} = (A \cup B \cup C, \chi(i_3), \lambda')$, where we insist that $V(\mathbf{A}) \cap V(\mathbf{G}_{i_1}) = Y = \chi(i_1)$ and we pick λ' as some vertex labelling of $A \cup B \cup C$ such that $\lambda|_{\chi(i_3)} \subseteq \lambda'$.

Similarly, if $\mathbf{A} = (A, Y, \lambda)$ is some t_{i_1}-boundaried graph, we denote $\mathbf{A}\overline{\mathbf{C}} = (A \cup C, \chi(i_3), \lambda')$, where we insist that $V(\mathbf{A}) \cap V(\mathbf{G}_{i_2}) = Y = \chi(i_2)$ and we pick λ' as some vertex labelling of $A \cup B \cup C$ such that $\lambda|_{\chi(i_3)} \subseteq \lambda'$.

We also set $ABCD = \mathbf{A} \oplus \overline{\mathbf{B}}\mathbf{CD}$, $ACD = \mathbf{A} \oplus \overline{\mathbf{C}}\mathbf{D}$, $ABCE = \mathbf{A} \oplus \overline{\mathbf{B}}\mathbf{CE}$, and $ACE = \mathbf{A}\overline{\mathbf{C}} \oplus \mathbf{E}$. Clearly:

$$\begin{aligned} ABCD &= \mathbf{AB}\overline{\mathbf{C}} \oplus \mathbf{D} && (7.18)\\ ACD &= \mathbf{A}\overline{\mathbf{C}} \oplus \mathbf{D} && (7.19)\\ ABCE &= \mathbf{AB}\overline{\mathbf{C}} \oplus \mathbf{E} && (7.20)\\ ACE &= \mathbf{A}\overline{\mathbf{C}} \oplus \mathbf{E} && (7.21) \end{aligned}$$

Finally we set $t = t_{i_1} = t_{i_2}, t' = t_{i_3} = |X_{\text{new}}|, c_{1,2} = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_{i_1}, \mathbf{G}_{i_2})$, and $c_{3,4} = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_{i_3}, \mathbf{G}_{\text{new}})$.

The proof is based on the following observations:

$\mathbf{G}_{i_1} \equiv_{\mathbf{p},t} \mathbf{G}_{i_2}$ means that

$$\begin{aligned} &\forall \mathbf{A} \in \mathcal{B}^{(t)}, \quad \mathbf{p}(\mathbf{A} \oplus \mathbf{G}_{i_1}) = \mathbf{p}(\mathbf{A} \oplus \mathbf{G}_{i_2}) + c_{1,2} \iff \\ &\forall \mathbf{A} \in \mathcal{B}^{(t)}, \quad \mathbf{p}(ABCD) = \mathbf{p}(ACD) + c_{1,2} \end{aligned} \tag{7.22}$$

$\mathbf{G}_{i_3} \equiv_{\mathbf{p},t'} \mathbf{G}_{\text{new}}$ means that

$$\begin{aligned} &\forall \mathbf{S} \in \mathcal{B}^{(t')}, \quad \mathbf{p}(\mathbf{S} \oplus \mathbf{G}_{i_3}) = \mathbf{p}(\mathbf{S} \oplus \mathbf{G}_{\text{new}}) + c_{3,4} \iff \\ &\forall \mathbf{S} \in \mathcal{B}^{(t')}, \quad \mathbf{p}(\mathbf{S} \oplus \mathbf{D}) = \mathbf{p}(\mathbf{S} \oplus \mathbf{E}) + c_{3,4} \end{aligned} \tag{7.23}$$

We apply (7.23) for all $\mathbf{S} \in \{\mathbf{AB}\overline{\mathbf{C}} \mid \mathbf{A} \in \mathcal{B}^{(t')}\}$,

$$\begin{aligned} &\forall \mathbf{A} \in \mathcal{B}^{(t)}, \quad \mathbf{p}(\mathbf{AB}\overline{\mathbf{C}} \oplus \mathbf{D}) = \mathbf{p}(\mathbf{AB}\overline{\mathbf{C}} \oplus \mathbf{E}) + c_{3,4} \overset{(7.18),(7.20)}{\iff} \\ &\forall \mathbf{A} \in \mathcal{B}^{(t)}, \quad \mathbf{p}(ABCD) = \mathbf{p}(ABCE) + c_{3,4} \end{aligned} \tag{7.24}$$

From (7.22) and (7.24), we obtain that

$$\forall \mathbf{A} \in \mathcal{B}^{(t)}, \quad \mathbf{p}(ABCE) = \mathbf{p}(ACD) + c_{1,2} - c_{3,4} \tag{7.25}$$

We apply (7.23) for all $\mathbf{S} \in \{(\mathbf{A}\overline{\mathbf{C}} \mid \mathbf{A} \in \mathcal{B}^{(t')}\}$,

$$\begin{aligned} &\forall \mathbf{A} \in \mathcal{B}^{(t)}, \qquad \mathbf{p}(\mathbf{A}\overline{\mathbf{C}} \oplus \mathbf{D}) = \mathbf{p}(\mathbf{A}\overline{\mathbf{C}} \oplus \mathbf{E}) + c_{3,4} \overset{(7.19),(7.21)}{\iff} \\ &\forall \mathbf{A} \in \mathcal{B}^{(t)}, \qquad \mathbf{p}(ACD) = \mathbf{p}(ACE) + c_{3,4} \end{aligned} \tag{7.26}$$

From (7.25) and (7.26) we conclude that

$$\begin{aligned} &\forall \mathbf{A} \in \mathcal{B}^{(t)}, \qquad \mathbf{p}(ABCE) = \mathbf{p}(ACE) + c_{1,2} \overset{(7.20),(7.21)}{\iff} \\ &\forall \mathbf{A} \in \mathcal{B}^{(t)}, \qquad \mathbf{p}(\mathbf{A} \oplus \overline{\mathbf{B}}\mathbf{C}\mathbf{E}) = \mathbf{p}(\mathbf{A} \oplus \overline{\mathbf{C}}\mathbf{E}) + c_{1,2} \end{aligned}$$

which implies that $\overline{\mathbf{B}}\mathbf{C}\mathbf{E} \equiv_{\mathbf{p},t'} \overline{\mathbf{C}}\mathbf{E}$ and $\mathsf{transp}_{\mathbf{p}}(\overline{\mathbf{B}}\mathbf{C}\mathbf{E}, \overline{\mathbf{C}}\mathbf{E}) = c_{1,2}$, as required.

Notice now that (7.26) can be rewritten as:

$$\forall \mathbf{A} \in \mathcal{B}^{(t)}, \qquad \mathbf{p}(\mathbf{A} \oplus \overline{\mathbf{C}}\mathbf{D}) = \mathbf{p}(\mathbf{A} \oplus \overline{\mathbf{C}}\mathbf{E}) + c_{3,4} \tag{7.27}$$

From (7.27) we obtain that $\overline{\mathbf{C}}\mathbf{D} \equiv_{\mathbf{p},t} \overline{\mathbf{C}}\mathbf{E}$ and $\mathsf{transp}_{\mathbf{p}}(\overline{\mathbf{C}}\mathbf{D}, \overline{\mathbf{C}}\mathbf{E}) = c_{3,4}$. Also, (7.23) implies that $\mathsf{transp}_{\mathbf{p}}(\mathbf{D}, \mathbf{E}) = c_{3,4}$. Therefore $\mathsf{transp}_{\mathbf{p}}(\overline{\mathbf{C}}\mathbf{D}, \overline{\mathbf{C}}\mathbf{E}) = \mathsf{transp}_{\mathbf{p}}(\mathbf{D}, \mathbf{E})$, thus $\mathbf{G}_{i_2} \equiv_{\mathbf{p},t} \mathbf{G}'_{i_2}$ and $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}_{i_2}, \mathbf{G}'_{i_2}) = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_{i_3}, \mathbf{G}_{\text{new}})$. □

7.6 The Algorithm

We reached are penultimate milestone. Here, we present the bits and parts of the algorithm in the proof of Theorem 7.3.1.

This algorithm uses two main subroutines: The algorithm of Lemma 7.6.9 and the algorithm of Lemma 7.6.15. The former detects a (β, f)-*rich protrusion* of G for some β and f that depend on $\mathbf{p}$. This detection requires $O(n^{2\beta})$ guesses for the root Y. For each such guess, we group together to a single set R all connected components with boundary Y and treewidth at most β and then we check whether $|R| > \mathbf{p}(G[R]) \cdot f(\beta)$. The

computation of $\mathbf{p}(G[R])$ can be done in $O(n)$ steps using the FII property (Lemma 7.6.6). Therefore we can detect such a rich protrusion, if one exists, in $O(n^{2\beta+1})$ steps.

7.6.1 Finding a transition pair

The type of pairs we want to compress are *null transition pairs*. We will postpone the definition of null transition pairs until Section 7.6.4, as we must first present some general technics.

Definition 7.6.1. Let $\mathbf{p}$ be a graph parameter that has FII. Let also $\mathbf{G}$ be a boundaried graph and let $D = (T, \chi, r)$ be a (α, β)-rooted tree-decomposition of $\mathbf{G}$. We say that a vertical pair (a, b) of (T, r) is a *transition pair* of $(\mathbf{G}, D)$ if (a, b) is $\equiv_{\mathbf{p},t}$-admissible and $\mathbf{G}_b \leq_{\mathrm{m}} \mathbf{G}_a$.

As an important step towards the proof of Theorem 7.3.1, we show that if a vertical pair (a, b) has "big enough" capacity, then we can rearrange the part of D that corresponds to the "inner territory" of (a, b) so that it now contains some transition pair (a', b') (Lemma 7.6.4). The proof of this result makes extensive use of some suitable variants of the concepts of lean and linked tree decompositions introduced in [32, 60].

Lemma 7.6.1. Let $\mathbf{p}$ be a graph parameter that has FII and let $t \in \mathbb{N}$. Let also $\mathbf{G}$ be a boundaried graph, $D = (T, \chi, r)$ be a binary tree-decomposition of $\mathbf{G}$ – of width at most $t - 1$ – and (a, b) be a transition pair of $(\mathbf{G}, D)$. If $(\mathbf{G}', D')$ is the (a, b)-compression of the pair $(\mathbf{G}, D)$ then $\mathbf{G}' \leq_{\mathrm{m}} \mathbf{G}$, $|\mathbf{G}'| < |\mathbf{G}|$, $\mathbf{G}' \equiv_{\mathbf{p},t} \mathbf{G}$ and $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}') = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_a, \mathbf{G}_b)$.

Proof. The fact that $\mathbf{G}' \leq_{\mathrm{m}} \mathbf{G}$ follows from the fact that $\mathbf{G}_a$ and $\mathbf{G}_b$ are compatible, thus, the corresponding frontier graphs H_a and H_b are equal, and the fact that $\mathbf{G}_b \leq_{\mathrm{m}} \mathbf{G}_a$.

Moreover, $|\mathbf{G}'| < |\mathbf{G}|$ follows from the definition of a binary tree-decomposition.

To prove that $\mathbf{G}' \equiv_{\mathbf{p},t} \mathbf{G}$ and that $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}') = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_a, \mathbf{G}_b)$, it is enough to apply Lemma 7.5.12 for $\mathbf{G}$ and D, by setting $i_1 = r, i_2 = r$, $i_3 = a$, and $\mathbf{G}_{\text{new}} = \mathbf{G}_b$. □

We declared in Theorem 7.3.1 that the graph obtained after the compression will be a minor of the input graph. To achieve this, we need to make sure that the part of the graph to be compressed will contain some vertex disjoint paths that can be contracted and yield a minor. We introduce the notions of *Weak leanness* and *Linkedness* of a tree-decomposition, that make a tree-decomposition locally lean, and, therefore, provide us with these disjoint paths.

Definition 7.6.2. Let $D = (T, \chi, r)$ be a tree-decomposition of a boundaried graph $\mathbf{G} = (G, X, \lambda)$. Given a vertical pair (a, b) of T, we say that D is *weakly* (a, b)*-lean for* $\mathbf{G}$ if for every $x, y \in \mathsf{inner}_{T,r}(a, b)$, such that x and y are (a, b)-aligned, and every $s \in \mathbb{N}$, it holds that

- either there are s vertex disjoint paths between $\chi(x)$ and $\chi(y)$ in G,
- or there is an edge $\{q, q'\} \in E(xTy)$ such that $|\chi(q) \cap \chi(q')| < s$.

Definition 7.6.3. Let $D = (T, \chi, r)$ be a tree-decomposition of a boundaried graph $\mathbf{G} = (G, X, \lambda)$ and (a, b) a vertical pair of T. D is (a, b)*-linked* if for every $x, y \in \mathsf{inner}_{T,r}(a, b)$, such that x and y are (a, b)-aligned, and every $s \in \mathbb{N}$, it holds that

- either there are s vertex disjoint paths between $\chi(x)$ and $\chi(y)$ in G,
- or there is an vertex $w \in V(xTy)$ such that $|\chi(w)| < s$.

Lemma 7.6.2. Let $\mathbf{G} = (G, A, \lambda)$ be a boundaried graph where $G[A]$ is a clique, let $B \subseteq V(G)$ where $G[B]$ is a clique, and let $D = (T, \chi, r)$ be a lean binary tree-decomposition of G such that T has a leaf l where $\chi(l) = B$. Let also G' be a graph obtained by removing from G edges that belong in $E(G[A])$ or in $E(G[B])$. Then D is weakly (r, l)-lean for $\mathbf{G}' = (G', A, \lambda)$.

Proof. Let $P = rTl$ and let $T_1, \ldots, T_z$ be the connected components of $T \setminus E(P)$. Recall that if x and y are (r, l)-aligned, then they both belong to one of $P, T_1, \ldots, T_z$. In any case, we have to prove that, if $\chi(x)$ and $\chi(y)$ are joined by a collection $\mathcal{Q} = \{Q_1, \ldots, Q_z\}$ of pairwise vertex disjoint paths of G, then they are also joined by the same collection of paths in G'. We choose $\mathcal{Q}$, so that $V(\cup\mathcal{Q})$ is minimized. This minimization forces each internal vertex of a path in $\mathcal{Q}$ to belong in $\chi(h)$ for some $h \in V(xTy)$ but not in $\chi(x) \cup \chi(y)$.

Suppose that some edge $e = \{i, j\}$ of some path $Q \in \{Q_1, \ldots, Q_z\}$ does not exist in G'. Clearly, the endpoints of e belong in one, say A, of A and B. Also, w.l.o.g., we assume that i is the vertex of $\{i, j\}$ that is closer to r in T and, again w.l.o.g., that i is an endpoint of $\{i, j\}$ that does not belong in $\chi(x)$ (by the choice of $\mathcal{Q}$ it is not possible that both i and j belong in $\chi(x)$ or to $\chi(y)$). We now claim that $i \in \chi(h)$ for some $h \in V(xTy) \setminus \{x\}$. Indeed, as we already mentioned, this holds when i is an internal vertex of Q while if i is not an internal vertex, then i should belong in $\chi(y)$. We conclude that i belongs both in $\chi(r)$ and $\chi(y)$ and as $x \in V(rTy), i \in \chi(x)$, a contradiction to the choice of i. Therefore all edges of the paths in $\mathcal{Q}$ are also edges of G'. □

The algorithm in the next lemma can make the subtree of a tree-decomposition between the vertices of a vertical pair linked.

Lemma 7.6.3. For every $t \in \mathbb{N}$, there is an algorithm with the following specifications:

	Pair Linking Algorithm
Input:	A quadruple $(\mathbf{G}, D, a, b)$ where: 1. $\mathbf{G}$ is a boundaried graph, 2. $D = (T, \chi, r)$ is a binary tree-decomposition of $\mathbf{G}$ of width at most $t - 1$, and 3. (a, b) is a vertical pair of (T, r).

Output: A binary tree-decomposition $D' = (T', \chi', r)$ where:
1. (a, b) is a vertical pair of (T', r),
2. $\mathsf{outer}_{T',r}(a, b) = \mathsf{outer}_{(T,r)}(a, b)$,
3. $\chi|_{\mathsf{outer}_{T',r}(a,b)} = \chi|_{\mathsf{outer}_{(T,r)}(a,b)}$,
4. D' is (a, b)-linked, and
5. $\mathsf{capacity}_{(T',r)}(a, b) \geq \frac{1}{4t} \cdot \mathsf{capacity}_{T,r}(a, b)$.

Moreover, this algorithm runs in $O_{t,c}(1)$ steps where $c = \mathsf{capacity}_{T,r}(a, b)$.

Proof. Let $I = \mathsf{inner}_{T,r}(a, b)$ and keep in mind that $\mathsf{capacity}_{T,r}(a, b) = |I|$. Let $A = \chi(a)$ and $B = \chi(b)$. Let $J = G[\bigcup_{x \in I} \chi(x)]$ and let $\overline{J}$ be the graph obtained from J if we make all pairs of vertices in A adjacent and all pairs of vertices in B adjacent, i.e., both A and B induce cliques in $\overline{J}$. Let $\overline{\mathbf{J}} = (\overline{J}, A, \lambda|_I)$. Let also $D^* = (T^*, \chi^*, a)$, where $T^* = T[I]$ and $\chi^* = \chi|_I$. Observe that D^* is a binary tree-decomposition of $\overline{\mathbf{J}}$ of width $\leq t - 1$. From Lemma 7.5.5, $|J| = |\overline{J}| \leq t \cdot |I| \leq t \cdot c$, where $c = \mathsf{capacity}_{T,r}(a, b)$.

From Lemma 7.5.8 there exists a lean binary tree-decompositions $D^\bullet = (T^\bullet, \chi^\bullet, a)$ of $\overline{\mathbf{J}}$, with width at most $t - 1$, such that $\chi^\bullet(a) = A$ and $|T^\bullet| \leq 2 \cdot |\overline{\mathbf{J}}| \leq 2 \cdot t \cdot c$. Notice that $D^\bullet$ can be found in $O_{|\overline{\mathbf{J}}|}(1) = O_{t,c}(1)$ steps.

We name b the vertex of $T^\bullet$ that has the biggest possible depth and $B \subseteq \chi(b)$. Our target is to modify $D^\bullet$ so that b will be a leaf of $T^\bullet$ and $\chi^\bullet(b) = B$. To this aim, we assume that b is not a leaf of $T^\bullet$ and apply modifications according to the following case analysis:

Case 1: $B \subsetneq \chi(b)$ and b has at most one child in $T^\bullet$, rename b to b_{old} in $D^\bullet$ and add in $T^\bullet$ a new vertex b_{new} along with the edge $\{b_{\text{old}}, b_{\text{new}}\}$ (see Figure 7.8, Case 1). Finally, set $\chi^\bullet = \chi^\bullet \cup (b_{\text{new}}, B)$ and rename b_{new} to b. Notice that, as $B \subsetneq \chi^\bullet(b_{\text{old}})$, this modification creates again a lean binary tree-decomposition of $\overline{\mathbf{J}}$.

Case 2: If $B = \chi(b)$ and b has one child d in $T^\bullet$, we copy the transfor-

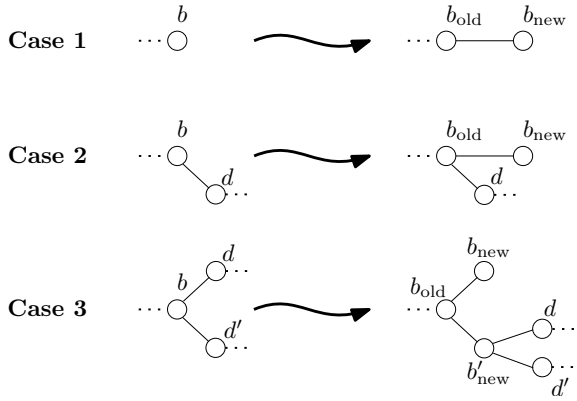


Figure 7.8: The transformations of Lemma 7.6.3.

mation of Case 1 (see Figure 7.8, Case 2). Notice that as $\chi(d)$ cannot be a subset of $\chi(b)$, the new decomposition $D^\bullet$ is again a lean binary decomposition of $\overline{\mathbf{J}}$.

Case 3: b has two children d and d' in $T^\bullet$. We remove the edges $\{b, d\}$ and $\{b, d'\}$ rename b to b_{old} in $D^\bullet$ and add two new vertices b_{new} and b'_{new}, and the edges $\{b_{\text{old}}, b_{\text{new}}\}$, $\{b'_{\text{new}}, d'\}$, $\{b_{\text{old}}, b'_{\text{new}}\}$, and $\{b'_{\text{new}}, d\}$ (see Figure 7.8, Case 3). We set $\chi^\bullet = \chi^\bullet \cup \{(b_{\text{new}}, B), (b'_{\text{new}}, \chi^\bullet(b))\}$and then we rename b_{new} to b. Observe that the fact that $\chi(d)$ is not a subset of $\chi(b)$ and $\chi(d')$ is not a subset of $\chi(b)$ in the original $D^\bullet$, the "new" $D^\bullet$ is again a lean binary decomposition of $\overline{\mathbf{J}}$.

After the above modifications, we have that b is indeed a leaf of $T^\bullet$. Also, these modifications may add at most 2 more nodes in $D^\bullet$. Now we are in position to apply Lemma 7.6.2 on $\overline{\mathbf{J}}$, B, $D^\bullet$, b, and J and obtain that $D^\bullet$ is a binary tree-decomposition that is weakly (a, b)-lean for $\mathbf{J} = (J, A, \lambda|_I)$.

We now construct $D' = (T', \chi', r)$ by setting $T' = (T \setminus I) \cup T^\bullet$, and $\chi' = \chi|_{V(G)\setminus I} \cup \chi^\bullet$. As b is a leaf of $T^\bullet$, D' is a binary tree-decomposition of $\mathbf{G}$ with width at most the width of D. Moreover, D' is a binary tree-decomposition that is weakly (a, b)-lean for $\mathbf{G}$.

We will transform D' to a tree-decomposition that is (a, b)-linked for $\mathbf{G}$ by applying the following transformation for every edge-pair (x, y) of $(T^\bullet, a)$:

If $|\chi'(x) \cap \chi'(y)| < \min\{|\chi'(x)|, |\chi'(y)|\}$, then remove $\{x, y\}$ from T', add a new vertex $v_{x,y}$ and the edges $\{x, v_{x,y}\}$ and $\{v_{x,y}, y\}$ and set $\chi' = \chi' \cup \{(v_{x,y}, \chi'(x) \cap \chi'(y))\}$.

This transformation makes D' (a, b)-linked while it does not harm the status of D' of being a binary tree-decomposition.

To prove the last property, first observe that $\mathsf{capacity}_{T,r}(a, b) = |T^*|$. As $D^* = (T^*, \chi^*, a)$ is a binary tree-decomposition of $\bar{\mathbf{J}}$, from Lemma 7.5.6 it holds that:

$$|T^*| \leq 4 \cdot |\bar{\mathbf{J}}| \Rightarrow |\bar{\mathbf{J}}| \geq |T^*|/4 \tag{7.28}$$

Now notice that $\mathsf{capacity}_{(T',r)}(a, b) \geq |T^\bullet|$, as we may add some vertices to T' when transforming $T^\bullet$ to an (a, b)-linked tree-decomposition. From Lemma 7.5.5, it holds that $|T^\bullet| \geq |\bar{\mathbf{J}}|/t$. Combining this with (7.28) we conclude that $|T^\bullet| \geq |T^*|/(4 \cdot t)$. □

Proposition 7.6.1 ([130,131]). *Let t, y be positive integers, and let $w \in \Sigma^*$, where Σ is an alphabet whose symbols are the numbers in $[t]$. If $|w| \geq y^t$, then there is a number $t' \in [t]$ and a subword u of w such that all letters in u are at least t' and u contains the number t' at least y times.*

The following Lemma give us an algorithm that finds a transition pair in a "locally linked" tree-decomposition of sufficient capacity.

Lemma 7.6.4. Let $\mathbf{p}$ be a graph parameter that is computable and has FII. For every $t \in \mathbb{N}$, there is an algorithm that receives as input a boundaried graph $\mathbf{G}$, a binary tree-decomposition $D = (T, \chi, r)$ of $\mathbf{G}$, and a vertical pair (a, b) of (T, r) such that

1. D has width at most $t - 1$,

2. $\mathsf{capacity}_{T,r}(a,b) \geq \mu_{\mathbf{p}}(t)$, and

3. D is (a,b)-linked,

and outputs a transition pair (a',b') for $(\mathbf{G},D)$ where $a',b' \in \mathsf{inner}_{T,r}(a,b)$. Moreover, this algorithm runs in $O_{t,c}(1)$ steps, where $c = \mathsf{height}_{T,r}(a)$.

Proof. We set $\theta = \theta_{\mathbf{p}}(t)$. From Lemma 7.5.3 and the definition of $\mu_{\mathbf{p}}$, there are $x,y \in \mathsf{inner}_{T,r}(a,b)$ such that x and y are (a,b)-aligned and $|xTy| \geq \theta = (\mathsf{card}_{\mathbf{p}}(t)\cdot t!+1)^{t+1}$. Let $P = xTy$ and keep in mind that all vertices of P belong in $\mathsf{inner}_{T,r}(a,b)$.

Let $\langle q_1,\ldots,q_\theta\rangle$ be the vertices of P ordered in the way they appear in P, starting from $q_1 = x$. We see the sequence $w = \langle|\chi(q_1)|,|\chi(q_1)|,\ldots,|\chi(q_\theta)|\rangle$ as a word on the alphabet $\Sigma = \{0\}\cup[t]$. As $\theta = (\mathsf{card}_{\mathbf{p}}(t)\cdot t!+1)^{t+1}$, from Proposition 7.6.1, there is some $t' \in \{0\}\cup[t]$, a pair $i',j' \in [\theta]$ and a set $I \subseteq \{i',i'+1,\ldots,j'\}$ such that

A. $|\chi(q_{i'})|,|\chi(q_{i'+1})|,\ldots,|\chi(q_{j'})| \geq t'$,

B. $|I| \geq \mathsf{card}_{\mathbf{p}}(t)\cdot t!+1$, and

C. for all $f \in I, |\chi(q_f)| = t'$.

Let $\sigma : \{\mathbf{G}_{q_h} \mid h \in I\} \to \mathcal{R}$ such that $\sigma(\mathbf{G}_{q_h}) = \mathsf{rep}_{\mathbf{p}}(\mathbf{G}_{q_h})$. As $|I| \geq \mathsf{card}_{\mathbf{p}}(t)\cdot t!+1$ and $t \geq t'$, there is a set $I' \subseteq I$ where $|I'| = t'!+1$ and such that all graphs in $\{\mathbf{G}_{q_h} \mid h \in I'\}$ are equivalent with respect to $\equiv_{\mathbf{p},t}$. Let $\mathcal{W} = \langle p_1,\ldots,p_{|I'|}\rangle$ be the vertices of I' ordered in the way they appear in P.

As D is (a,b)-linked, and because of **A** and **C**, for every $i,j \in [t'!+1]$, the vertices of $\chi(p_i)$ and the vertices of $\chi(p_j)$ are connected in $\mathbf{G}_{p_i}$ by a collection $\mathcal{Q}_{i,j}$ of t' internally vertex-disjoint paths. For every $i,j \in [t'!+1], i \leq j$, we define the bijection $\kappa_{i,j} : [t'] \to [t']$ such that for each $m \in [t]$, it holds that the vertices $\psi^{-1}_{\mathbf{G}_{p_i}}(m)$ and $\psi^{-1}_{\mathbf{G}_{p_j}}(m)$ are the two endpoints of some path in $\mathcal{Q}_{i,j}$ (recall that $\psi_{\mathbf{G}_{p_i}}$ and $\psi_{\mathbf{G}_{p_j}}$ are the label

normalising functions of $\mathbf{G}_{p_i}$ and $\mathbf{G}_{p_j}$ respectively). Notice that if q_i, q_j, and q_h are vertices in $\mathcal{W}$ where $i \leq j \leq h$, then, $\kappa_{i,h} = \kappa_{i,j} \circ \kappa_{j,h}$. As $[t']$ has $t'!$ different permutations, there exist $i, j \in [t!+1], i < j$ such that $\kappa_{i,j}$ is the identity mapping. This means that $\mathcal{Q}_{i,j}$ contains t' vertex-disjoint paths $P_1, \ldots, P_{t'}$ in $\mathbf{G}_{p_i}$, from $\chi(p_i)$ to $\chi(p_j)$, such that the endpoints of each path are equally indexed. Let $a' = p_i$ and $b' = p_j$. We assume that for $m \in [t']$, P_m is the path with endpoints $\psi^{-1}_{\mathbf{G}_{a'}}(m)$ and $\psi^{-1}_{\mathbf{G}_{b'}}(m)$. It remains to show that $\mathbf{G}_{b'} \leq_{\mathrm{m}} \mathbf{G}_{a'}$.

To prove this, we define the injective function $\mu : V(\mathbf{G}_{b'}) \to V(\mathbf{G}_{a'})$, where:

$$\mu(v) = \begin{cases} V(P_{\psi_{\mathbf{G}_{b'}}(v)}) & \text{if } v \in \chi(b') \\ v & \text{if } v \in \tilde{V}_{b'} = V_{b'} \setminus \chi(b') \end{cases}$$

Intuitively, $\mathbf{G}_{b'}$ can be obtained from $\mathbf{G}_{a'}$ if we remove all vertices in $V_{a'} \setminus V_{b'}$ that do not belong in some of the paths in $\mathcal{Q}_{i,j} = \{P_1, \ldots, P_{t'}\}$ and then, for each $h \in [t']$ contract the path P_h to its endpoint in $\chi(w)$, i.e., the vertex of $\chi(w)$ that has index h.

Notice that the pair (p_i, p_j) can be found algorithmically inside $\mathbf{G}_a$ and D_a, given that the function σ is computable and this is possible because of Lemma 7.5.4. □

The algorithm of Lemma 7.6.5 takes the transition pair the algorithm of Lemma 7.6.4 outputs and compresses it.

Lemma 7.6.5. Let **p** be a graph parameter that is computable and has FII. For every $t \in \mathbb{N}$, there is an algorithm with the following specifications:

	Pair Compression Algorithm
Input:	A quadruple $(\mathbf{G}, D, a, b)$ where: 1.$\mathbf{G}$ is a boundaried graph, 2. $D = (T, \chi, r)$ is a binary tree-decomposition of $\mathbf{G}$ of width at most $t - 1$, and

	3. (a, b) is a vertical pair of (T, r) where $4t \cdot \mu_{\mathbf{p}}(t) \leq \mathsf{capacity}_{T,r}(a, b)$.
Output:	A triple $(\mathbf{G}', D', \ell)$ where: 1. $\mathbf{G} \equiv_{\mathbf{p},t} \mathbf{G}'$, 2. $\lvert\mathbf{G}'\rvert < \lvert\mathbf{G}\rvert$, 3. $\mathbf{G}' \leq_{\mathrm{m}} \mathbf{G}$, 4. $D' = (T', \chi', r)$ is a binary tree-decomposition of $\mathbf{G}'$ of width at most $t - 1$, and 5. the value $\ell = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}')$.

Moreover, if (x, y) is a transition pair where $(x, y) \in \mathsf{outer}_{T,r}(a, b)$, then (x, y) is also a transition pair of $(\mathbf{G}', D')$ and $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}_x, \mathbf{G}_y) = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}'_x, \mathbf{G}'_y)$. This algorithm runs in $O_{t,c}(1)$ steps, where $c = \mathsf{height}_{T,r}(a)$.

Proof. The algorithm applies first the algorithm of Lemma 7.6.3 on $(\mathbf{G}, D, a, b)$ and constructs a binary tree-decomposition $D^* = (T^*, \chi^*, r)$ with the specifications of Lemma 7.6.3. Notice that

$$\mathsf{capacity}_{(T^*,r)}(a, b) \geq \frac{1}{4t} \cdot \mathsf{capacity}_{T,r}(a, b) \geq \mu_{\mathbf{p}}(t).$$

Next applies the algorithm of Lemma 7.6.4 in order to find a transition pair (a', b') of $(\mathbf{G}, D^*)$, where $a', b' \in \mathsf{inner}_{(T^*,r)}(a, b)$. Let $(\mathbf{G}', D')$ be the (a', b')-compression of the pair $(\mathbf{G}, D^*)$. By Lemma 7.6.1, $\mathbf{G}' \leq_{\mathrm{m}} \mathbf{G}$, $\lvert\mathbf{G}'\rvert < \lvert\mathbf{G}\rvert$, $\mathbf{G}' \equiv_{\mathbf{p},t} \mathbf{G}$, and $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}') = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_{a'}, \mathbf{G}_{b'})$. Notice that

$$\mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}') = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_{a'}, \mathbf{G}'_{b'}) = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_{a'}, \mathbf{H}) + \mathsf{transp}_{\mathbf{p}}(\mathbf{H}, \mathbf{G}_{b'}),$$

where $\mathbf{H} = \mathsf{rep}_{\mathbf{p}}(\mathbf{G}_{a'}) = \mathsf{rep}_{\mathbf{p}}(\mathbf{G}'_{b'})$. We now use Lemma 7.5.4 to compute $\mathbf{H}$, $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}_{a'}, \mathbf{H})$ and $\mathsf{transp}_{\mathbf{p}}(\mathbf{H}, \mathbf{G}_{b'})$ in $O_{t,c}(1)$ steps. Therefore, $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}')$ can be computed in $O_{t,c}(1)$ steps.

The last statement of the lemma is straightforward in the case where

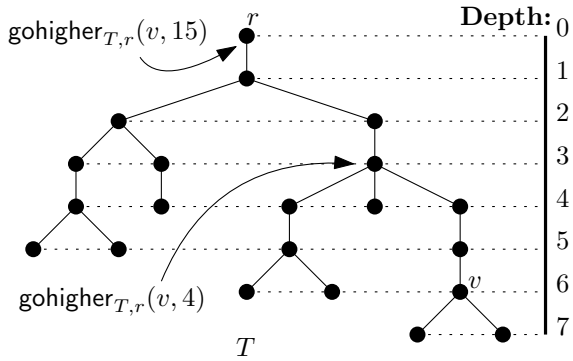


Figure 7.9: An example of $\mathsf{gohigher}_{T,r}(v, \mu)$.

$a \notin \mathsf{desc}_{(T,t)}(y)$ as, in this case, $\mathbf{G}_x = \mathbf{G}'_x$. Suppose now that $a \in \mathsf{desc}_{(T,t)}(y)$. We apply Lemma 7.5.12 on $\mathbf{G}$ and D' for $i_1 = x$, $i_2 = y$, $i_3 = a'$ and $\mathbf{G}_{\text{new}} = \mathbf{G}_{b'}$ and we obtain that $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}_x, \mathbf{G}_y) = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}'_x, \mathbf{G}'_y)$ as required. □

7.6.2 A boundaried graph compression algorithm

Definition 7.6.4. Let (T, r) be a rooted tree. If $v \in (T)$ and $\mu \in \mathbb{N}$, we denote by $\mathsf{gohigher}_{T,r}(v, \mu)$ the unique vertex u of T where

- $v \in \mathsf{desc}_{T,r}(u)$ and
- $\min\{\mu, \mathsf{depth}_{T,r}(v) - \mathsf{depth}_{T,r}(u)\}$ is maximized.

(See Figure 7.9 for an example.)

The next algorithm is a big step towards proving Theorem 7.3.1. It can compress a boundaried graph to a size depending on the width of the decomposition we are given as input, and the number of the equivalence classes of $\equiv_{\mathbf{p},t}$. This algorithm runs in time proportional to the size of our graph.

Lemma 7.6.6. Let **p** be graph parameter that is computable and has FII. For every $t \in \mathbb{N}$, there is an algorithm with the following specifications:

	Boundaried Graph Compression Algorithm
Input:	A pair $(\mathbf{G}, D)$ where: 1.$\mathbf{G} = (G, X, \lambda)$ is a boundaried graph, and 2. $D = (T, \chi, r)$ is a binary tree-decomposition of $\mathbf{G}$ of width at most $t-1$.
Output:	A pair $(\mathbf{G}', z)$ where: 1. $\mathbf{G} \equiv_{\mathbf{p},t} \mathbf{G}'$, 2. $\mathbf{G}' \leq_{\mathrm{m}} \mathbf{G}$, and 3. $\lvert\mathbf{G}'\rvert \leq t \cdot (2^{4t \cdot \mu_{\mathbf{p}}(t)-1} - 1)$, 4. $\mathbf{tw}(\mathbf{G}') \leq t-1$, and 5. the value $z = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}')$.

Moreover, the running time of this algorithm is $O_t(|\mathbf{G}|)$.

Proof. The algorithm is the following:

Step 1: First we set some variables:

- $\hat{\mathbf{G}} = \mathbf{G}$,
- $\hat{D} = D$, and
- $z = 0$ (at the end this variable will be equal to $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}')$).

Let $\hat{D} = (\hat{T}, \hat{\chi}, r)$, and $\hat{\mathbf{G}} = (\hat{G}, X, \hat{\lambda})$. Let also $\mu = 4t \cdot \mu_{\mathbf{p}}(t)$.

Step 2: Let y be a leaf of $\hat{T}$ with maximum height and let $w = \mathsf{gohigher}_{(\hat{T},r)}(y, \mu)$.

Step 3: If $\mathsf{height}_{(\hat{T},r)}(w) \leq \mu - 1$, i.e., the height of $\hat{T}$ is at most $\mu - 1$, then output $\mathbf{G}' = \hat{\mathbf{G}}$ and z and **stop**. As $\hat{T}$ is a binary tree, we have that $|\hat{T}| \leq 2^{(\mu-1)} - 1$, therefore, from Lemma 7.5.5, $|\mathbf{G}'| \leq t \cdot (2^{(\mu-1)} - 1)$.

Step 4: If $\mathsf{height}_{(\hat{T},r)}(w) = \mu$, then notice that $|\hat{T}_w| \leq 2^\mu - 1$, where $\hat{T}_w = \hat{T}[\mathsf{desc}_{\hat{T},r}(w)]$. Thus, from Lemma 7.5.5, $|\hat{\mathbf{G}}_w| \leq t \cdot (2^\mu - 1)$. Let l be a leaf of $\hat{T}_w$ that is in distance $\mu - 1$ from w in $\hat{T}_w$ and notice that the vertical pair (w, l) has capacity at least μ and at most $2^\mu - 1$. Let $(\hat{G}', \hat{D}', \ell)$ be the output of the algorithm of Lemma 7.6.5 when it runs with input $(\hat{\mathbf{G}}, \hat{D}, w, l)$. As the part of $\hat{D}$ that has changed in $\hat{D}'$ concerns only vertices of $\hat{\mathbf{G}}_w$, the computation of $(\hat{G}', \hat{D}', \ell)$ can be done in $O_t(1)$ steps. Keep also in mind that $\hat{\mathbf{G}}' \leq_{\mathrm{m}} \hat{\mathbf{G}}$, $|\hat{\mathbf{G}}'| < |\hat{\mathbf{G}}|$, $\hat{\mathbf{G}} \equiv_{\mathbf{p},t} \hat{\mathbf{G}}'$ and $\ell = \mathsf{transp}_{\mathbf{p}}(\hat{\mathbf{G}}, \hat{\mathbf{G}}')$.

Step 5: Set $\hat{\mathbf{G}} = \hat{\mathbf{G}}'$, $\hat{D} = \hat{D}'$, and $z = z + \ell$, and go to **Step 2**.

Notice that each of the steps of the algorithm above is repeated less than $|\mathbf{G}|$ times and that it returns a pair $(\mathbf{G}', z)$ where $\mathbf{G}' \leq \mathbf{G}, \mathbf{G} \equiv_{\mathbf{p},t} \mathbf{G}'$, $|\mathbf{G}'| \leq t \cdot (2^\mu - 1)$, $\mathbf{tw}(\mathbf{G}') \leq t - 1$, and $z = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}')$. As each step takes $O_t(1)$ steps, the algorithm runs in $O_t(|\mathbf{G}|)$ steps. □

We can use the *Boundaried Graph Compression Algorithm* to device a linear algorithm computing the value of a (computable) parameter $\mathbf{p}$ that has FII, for boundaried graphs. The hidden constants depend on the treewidth and the number of the equivalence classes of $\equiv_{\mathbf{p},t}$.

Lemma 7.6.7. Let $\mathbf{p}$ be a graph parameter that has FII and let $t \in \mathbb{N}$. Let $\mathbf{G}, \mathbf{G}' \in \mathcal{T}^{(\leq t)}$, where $\mathbf{G} \equiv_{\mathbf{p},t} \mathbf{G}'$. Then $\mathbf{p}(G) = \mathbf{p}(G') + \mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}')$ where G and G' are the underlying graphs of $\mathbf{G}$ and $\mathbf{G}'$.

Proof. Let t' be the boundary size of $\mathbf{G}$ and $\mathbf{G}'$. We denote $\mathbf{G} = (G, X, \lambda)$. The fact that $\mathbf{G} \equiv_{\mathbf{p},t} \mathbf{G}'$ implies that:

$$\forall \mathbf{F} \in \mathcal{B}^{(t')}, \quad \mathbf{p}(\mathbf{G} \oplus \mathbf{F}) - \mathbf{p}(\mathbf{G}' \oplus \mathbf{F}) = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}') \tag{7.29}$$

We apply (7.29) for $\mathbf{F} = \mathbf{B} = (G[X], X, \lambda_X)$. Notice that $\mathbf{G} \oplus \mathbf{B} = G$ and $\mathbf{G}' \oplus \mathbf{B} = G'$. Therefore $\mathbf{p}(G) = \mathbf{p}(G') + \mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}')$. □

LEMMA 7.6.8. Let $t \in \mathbb{N}$, $\mathbf{p}$ be a graph parameter that is computable and has FII. There is an algorithm that, given a $\mathbf{G} = (G, X, \lambda) \in \mathcal{T}^{(\leq t)}$, outputs $\mathbf{p}(G)$ in $O_t(|G|)$ steps.

Proof. First we compute a binary tree-decomposition $D = (T, \chi, r)$ of $\mathbf{G}$, with width at most $t-1$, in $O_t(|\mathbf{G}|)$ steps using the algorithm of Lemma 7.5.10. Then, by running the algorithm of Lemma 7.6.6, with input $\mathbf{G}$ and D, we obtain – in $O_t(|G|)$ steps – some $\mathbf{G}' \in \mathcal{T}^{(t)}$, where $\mathbf{G} \equiv_{\mathbf{p},t} \mathbf{G}'$, $|\mathbf{G}'| \leq t \cdot (2^{4t \cdot \mu_{\mathbf{p}}(t)-1} - 1)$, together with the value of $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}')$. From Lemma 7.6.7, $\mathbf{p}(G) = \mathbf{p}(G') + \mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}')$. As

$$|G'| \leq t \cdot (2^{4t \cdot \mu_{\mathbf{p}}(t)-1} - 1) = O_t(1),$$

$\mathbf{p}(G')$, and therefore $\mathbf{p}(G)$, can be computed in $O_t(1)$ steps. □

7.6.3 An algorithm for finding a rich protrusion

Let us formally define what a rich protrusion is.

DEFINITION 7.6.5. Let $\mathbf{p}$ be a graph parameter and $f : \mathbb{N} \to \mathbb{N}$ be some computable function. Let also $\beta \in \mathbb{N}$, a graph G along with a labeling $\lambda : V(G) \to \mathbb{N}$ of G, and $R \subseteq V(G)$. We denote $Y = \partial_G(R)$ and we say that R is a (β, f)-*rich protrusion* of G for $\mathbf{p}$ if:

(1) $|Y| \leq \beta$ and the boundaried graph $(G[R], Y, \lambda|_R)$ has treewidth at most β,

(2) $Y = N_G(R \setminus Y)$,

(3) the set R is maximal with respect to (1) and (2),

(4) $|R| > \mathbf{p}(G[R]) \cdot f(\beta)$.

Notice that, in the above definition, R is a β-protrusion of G.

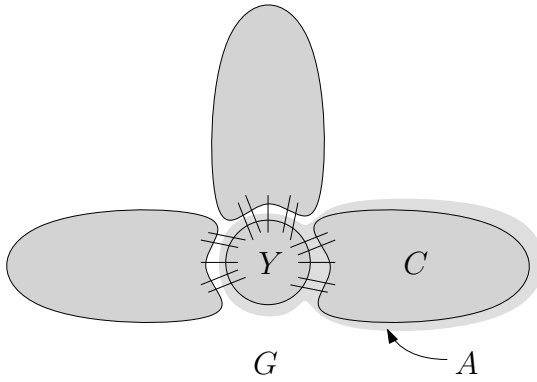


Figure 7.10: A is an *augmented connected component for* (G, Y).

Definition 7.6.6. Let G be a graph and $Y \subseteq V(G)$. We say that a graph A is an *augmented connected component* for (G, Y) if there is some connected component C of $G \setminus Y$ such that $A = G[Y \cup V(C)]$ (see Figure 7.10). We denote by $\mathcal{A}(G, Y)$ the set containing the vertex set of the augmented connected components for (G, Y).

The use of the algorithm of Lemma 7.6.9 is self-explanatory.

Lemma 7.6.9. Let **p** be a graph parameter that is computable and has FII. Let also $f : \mathbb{N} \to \mathbb{N}$ be a computable function. There is an algorithm with the following specifications:

	Rich Protrusion Finding Algorithm
Input:	A graph G and a $\beta \in \mathbb{N}$.
Output:	Either a (β, f)-rich protrusion of G for **p**, or reports that such a (β, f)-rich protrusion does not exist.

This algorithm runs in $O_\beta(|G|^{\beta+1})$ steps.

Proof. Let λ be a labelling of G. The algorithm considers the set $\mathcal{Y}$ of all possible subsets of $V(G)$ of at most β vertices. For every $Y \in \mathcal{Y}$, let

$\mathcal{A}_Y$ be the set containing every $R \in \mathcal{A}(G, Y)$ where $N_G(R \setminus Y) = Y$ and $\mathbf{tw}(G[R], Y, \lambda|_R) \leq \beta$ (to check whether the treewidth is at most β, we can use the algorithm of Lemma 7.5.10). Also, let $R_Y = \cup\mathcal{A}_Y$ for $Y \in \mathcal{Y}$. The collection $\mathcal{R} = \{R_Y, Y \in \mathcal{Y}\}$ can be constructed in $O_\beta(|G|^\beta)$ steps. Also, every (β, f)-rich protrusion of G for $\mathbf{p}$ should belong in $\mathcal{R}^- = \{R_Y \setminus Y, Y \in \mathcal{Y}\}$. The only thing remaining for the algorithm to check is whether for some $R \in \mathcal{R}$, $|R| > \mathbf{p}(G[R]) \cdot f(\beta)$. This can be done using Lemma 7.6.8. If such an R is found, it is a (β, f)-rich protrusion for $\mathbf{p}$, otherwise report that G has no (β, f)-rich protrusion for $\mathbf{p}$. □

Our next step is to prove that if G does not have rich protrusions, then $|G| = O(\mathbf{p}(G))$. For this, we already know that G has a $(O(\mathbf{p}(G)), 2\beta)$-rooted tree-decomposition (Lemma 7.5.11) and we prove that

$$\sum_{i \in [s]} \mathbf{p}(G_{r^{(i)}}) \leq \mathbf{p}(G) + O(1),$$

where $\mathsf{children}_{T,r}(r) = \{r^{(1)}, \ldots, r^{(s)}\}$ (Lemma 7.6.10). Notice that as G does not contain any rich protrusion and, as each $\mathbf{G}_{r^{(i)}}$ can be seen as such a protrusion, we have that $|G_{r^{(i)}}| = O(\mathbf{p}(G_{r^{(i)}}))$. This, together with the previous inequality and the fact that $|G| = \sum_{i \in [s]} |G_{r^{(i)}}| + O(\mathbf{p}(G))$ imply that $|G| = O(\mathbf{p}(G))$ (see Lemma 7.6.11). We conclude that if G does not have rich protrusions, its size is already linear on the value of the parameter $\mathbf{p}$ on G.

Lemma 7.6.10. Let $\mathbf{p}$ be a graph parameter that has FII. Let also $\mathbf{G} = (G, X, \lambda)$ be boundaried graph and let $D = (T, \chi, r)$ be an (α, β)-tree-decomposition of $\mathbf{G}$. If $\mathsf{children}_{T,r}(r) = \{r^{(1)}, \ldots, r^{(\ell)}\}$, then

$$\sum_{i \in [\ell]} \mathbf{p}(G_{r^{(i)}}) \leq \mathbf{p}(G) + \alpha \cdot \xi_{\mathbf{p}}(\beta).$$

Proof. From Lemma 7.6.6, for every $i \in [\ell]$, there is a graph $\mathbf{G}'_{r^{(i)}}$ where

$\mathbf{G}_{r^{(i)}} \equiv_{\mathbf{p},\beta} \mathbf{G}'_{r^{(i)}}$, $\mathbf{G}'_{r^{(i)}} \leq_{\mathrm{m}} \mathbf{G}_{r^{(i)}}$, and $|\mathbf{G}'_{r^{(i)}}| \leq m$, where $m = \beta \cdot (2^{4\beta \cdot \mu_{\mathbf{p}}(\beta)-1} - 1)$. We also set $l_i = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_{r^{(i)}}, \mathbf{G}'_{r^{(i)}})$, $i \in [\ell]$. From Lemma 7.6.7

$$\mathbf{p}(\mathbf{G}_{r^{(i)}}) = l_i + \mathbf{p}(\mathbf{G}'_{r^{(i)}}) \leq l_i + \tau_{\mathbf{p}}(|\mathbf{G}'_{r^{(i)}}|) \leq l_i + \tau_{\mathbf{p}}(m),$$

therefore:

$$\begin{aligned}
\sum_{i\in[\ell]} \mathbf{p}(G_{r^{(i)}}) &\leq \sum_{i\in[\ell]} (l_i + \tau_{\mathbf{p}}(m)) \\
&= (\sum_{i\in[\ell]} l_i) + \ell \cdot \tau_{\mathbf{p}}(m) \\
&\leq (\sum_{i\in[\ell]} l_i) + \alpha \cdot \tau_{\mathbf{p}}(m) \\
&= (\sum_{i\in[\ell]} l_i) + \alpha \cdot \xi_{\mathbf{p}}(\beta) \qquad (7.30)
\end{aligned}$$

It holds that for every $i \in [\ell]$:

$$\forall \mathbf{F} \in \mathcal{B}^{(t_{r^{(i)}})}, \quad \mathbf{p}(\mathbf{G}_{r^{(i)}} \oplus \mathbf{F}) - \mathbf{p}(\mathbf{G}'_{r^{(i)}} \oplus \mathbf{F}) = l_i \qquad (7.31)$$

Let $G^{(0)} = G$. We set:

$$\begin{aligned}
\mathbf{F}_0 &= (G^{(0)}, \chi(r^0), \lambda|_{G^{(0)}}) \setminus \tilde{V}_{r^{(0)}}, & G^{(1)} &= \mathbf{F}_0 \oplus \mathbf{G}'_{r^{(0)}} \\
\mathbf{F}_1 &= (G^{(1)}, \chi(r^{(1)}), \lambda|_{G^{(1)}}) \setminus \tilde{V}_{r^{(1)}}, & G^{(2)} &= \mathbf{F}_1 \oplus \mathbf{G}'_{r^{(1)}} \\
&\ldots \\
\mathbf{F}_{\ell-1} &= (G^{(\ell-1)}, \chi(r^{(\ell-1)}), \lambda|_{G^{(\ell-1)}}) \setminus \tilde{V}_{r^{(\ell-1)}}, & G^{(\ell)} &= \mathbf{F}_{\ell-1} \oplus \mathbf{G}'_{r^{(\ell-1)}}
\end{aligned}$$

If we repetitively apply (7.31) for $i \in [\ell]$ and $\mathbf{F} = \mathbf{F}_i$, we have that $\mathbf{p}(G^{(i)}) = \mathbf{p}(G^{(i-1)}) - l_i$. Combining these equalities altogether we con-

clude that

$$\mathbf{p}(G) - \sum_{i \in [\ell]} l_i = \mathbf{p}(G^{(\ell)}) \geq 0.$$

Therefore, $\sum_{i \in [\ell]} l_i \leq \mathbf{p}(G)$.

This, together with (7.30), completes the proof of the lemma. □

LEMMA 7.6.11. Let $\mathbf{p}$ be a graph parameter that has FII and is protrusion decomposable. Let also $c = \mathsf{dec}(\mathbf{p})$ and $f : \mathbb{N} \to \mathbb{N}$ be a computable function. For every graph G, if G does not contain any $(2c, f)$-rich protrusion W for $\mathbf{p}$, then $|G| \leq d \cdot \mathbf{p}(G)$, where $d = (f(2c) \cdot (c \cdot \xi_{\mathbf{p}}(2c) + 1) + c)$.

Proof. According to the definition of protrusion decomposability in (7.1), G has a $(c \cdot \mathbf{p}(G), c)$-protrusion decomposition. Let λ be a labeling of G. From Lemma 7.5.11 G is the underlying graph of some boundaried graph $\mathbf{G} = (G, X, \lambda)$ that has a $(c \cdot \mathbf{p}(G), 2c)$-tree-decomposition $D = (T, \chi, r)$.

We consider such a $(c \cdot \mathbf{p}(G), 2c)$-tree-decomposition D so that $\chi(r)$ is minimized. Let $\{r^{(1)}, \ldots, r^{(\ell)}\}$ be the set of children of r in (T, r). A consequence of the minimality of $\chi(r)$ and Conditions (4) and (5) of the definition of an (a, b)-tree-decomposition, is that property (3) in the definition of a $(2c, f)$-rich protrusion is satisfied for all $V_{r^{(i)}}, i \in [\ell]$.

As none of $V_{r^{(i)}}$ can be a $(2c, f)$-rich protrusion of G for $\mathbf{p}$, we deduce that:

$$\forall i \in [\ell], \quad |\mathbf{G}_{r^{(i)}}| \leq \mathbf{p}(G_{r^{(i)}}) \cdot f(t_{r^{(i)}}) \tag{7.32}$$

From Lemma 7.6.10 it holds that:

$$\begin{aligned} \sum_{i \in [\ell]} \mathbf{p}(G_{r^{(i)}}) &\leq \mathbf{p}(G) + c \cdot \mathbf{p}(G) \cdot \xi_{\mathbf{p}}(2c) \\ &= \mathbf{p}(G) \cdot (1 + c \cdot \xi_{\mathbf{p}}(2c)) \end{aligned} \tag{7.33}$$

Using (7.32) and (7.33) we deduce

$$\begin{aligned}
|G| &\leq |X| + \sum_{i\in[\ell]} |\mathbf{G}_{r^{(i)}}| \\
&\leq c\cdot \mathbf{p}(G) + f(2c)\cdot \sum_{i\in[\ell]} \mathbf{p}(G_{r^{(i)}}) \\
&\leq c\cdot \mathbf{p}(G) + f(2c)\cdot \mathbf{p}(G)\cdot(1 + c\cdot \xi_{\mathbf{p}}(2c)) \\
&\leq (f(2c)\cdot(c\cdot \xi_{\mathbf{p}}(2c)+1) + c)\cdot \mathbf{p}(G)
\end{aligned}$$

which gives the claimed upper bound. □

7.6.4 Finding and compressing a null-transition pair

What remains now is to deal with the case where a rich protrusion R is detected. Notice that R is the vertex set of some boundaried graph $\mathbf{H} = (H, Y, \lambda)$ that has a binary rooted tree decomposition $D = (T, \chi, r)$ of width $O(1)$. Following the methodology of [155], one could replace in G the boundaried graph $\mathbf{H}$ by a smaller "protrusion" $\mathbf{H}'$. This indeed gives a smaller equivalent instance (G', k'), however, we do not have any guaranty that G' will be a minor of G and that $k' = k$, as it is required for Theorem 7.3.1. In order to enforce these additional properties, we follow another approach, the algorithm of Lemma 7.6.15, that intuitively consists of "compressing" the protrusion than "replacing" it with something smaller.

Let us define the aforementioned null transition pair.

Definition 7.6.7. Let $\mathbf{G}$ be a boundaried graph and $D = (T, \chi, r)$ a (α, β)-tree-decomposition of it. The *potential* of a transition pair (a, b) of $(\mathbf{G}, D)$ is defined to be

$$\mathsf{potential}_{(\mathbf{G},D)}(a, b) = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_a, \mathbf{G}_b).$$

If the potential of a transition pair (a, b) is 0, the we say that (a, b) is a *null transition pair*.

Definition 7.6.8. A *transition collection* of $(\mathbf{G}, D)$ is a pair collection $\mathcal{P}$ of (T, r) consisting of transition pairs of $(\mathbf{G}, D)$. The *potential* of $\mathcal{P}$, denoted by $\mathsf{potential}_{(\mathbf{G},D)}(\mathcal{P})$, is the sum of the potentials of the pairs of $\mathcal{P}$.

Bounding potentials

Let $\mathbf{H}$ be a rich protrusion, the tree T in its tree decomposition is big enough to guarantee the existence of a pair collection $\mathcal{C}$ of more than $\mathbf{p}(G)$ transition pairs. For each such pair the value $\mathsf{transp}_{\mathbf{p}}(\mathbf{H}_a, \mathbf{H}_b)$ expresses how much the parameter k should be reduced while producing an equivalent instance because of an (a, b)-compression. Using the fact that $\mathbf{p}$ is a minor-closed parameter, we prove that this value is never negative (Lemma 7.6.13). Therefore the potential of $\mathcal{C}$ is also non-negative. Notice that this is the only point in the proof that we use the minor-closedness of $\mathbf{p}$.

We next prove that if we apply all (a, b)-compressions in $\mathcal{C}$ (in any order) the *total reduction* of the parameter in the produced equivalent instance will be at least the potential of $\mathcal{C}$ (the proof is strongly based on the fact that the pairs in $\mathcal{C}$ are mutually non-interfering). As this total reduction cannot be bigger than $\mathbf{p}(G)$ we conclude that at least one of the pair in $\mathcal{C}$ is a null-transition pair.

Lemma 7.6.12. Let $\mathbf{p}$ be a graph parameter that has FII and let $t \in \mathbb{N}$. Let also $\mathbf{G}$ be a boundaried graph and $D = (T, \chi, r)$ a binary tree-decomposition of $\mathbf{G}$ of width at most t. For every transition collection $\mathcal{P}$ of $(\mathbf{G}, D)$, it holds that $\mathsf{potential}_{(\mathbf{G},D)}(\mathcal{P}) \leq \mathbf{p}(G)$.

Proof. Notice that the lemma is obvious when $\mathsf{potential}_{(\mathbf{G},D)}(\mathcal{P}) = 0$.

Assume now that the lemma holds for every choice of $(\mathbf{G}, D)$ and $\mathcal{P}$ with $\mathsf{potential}_{(\mathbf{G},D)}(\mathcal{P}) < k$, where $k \in \mathbb{N}^+$. Let $\mathcal{P}$ be a transition collection of $(\mathbf{G}, D)$ where $\mathsf{potential}_{(\mathbf{G},D)}(\mathcal{P}) = k > 0$. Clearly, $\mathcal{P}$ contains some non-null-transition pair (a, b). We set $t = t_a = t_b$. Assume that $\ell = \mathsf{potential}_{(\mathbf{G},D)}(a, b) = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_a, \mathbf{G}_b)$ and keep in mind that $\ell > 0$. We set $\mathcal{P}' = \mathcal{P} \setminus \{(a, b)\}$.

Let $(\mathbf{G}', D')$ be the (a, b)-compression of $(\mathbf{G}, D)$.

Claim 4. $\mathsf{potential}_{(\mathbf{G}',D')}(\mathcal{P}') = k - \ell$.

Proof of Claim 4. We partition $\mathcal{P}$ in three sets. Let $\mathcal{P}_{\text{down}}$ contain all the pairs $(i, j) \in \mathcal{P}$ such that $i \in \mathsf{children}_{T,r}(b)$, let $\mathcal{P}_{\text{side}}$ contains all the pairs $(i, j) \in \mathcal{P}$ such that $i \neq_{T,r} a$ and let $\mathcal{P}_{\text{up}}$ contains all the pairs $(i, j) \in \mathcal{P}$ such that $i \geq_{T,r} a$. The fact that all pairs $\mathcal{P}$ are pairwise non-interfearing pairs of (T, r), implies that $\{\mathcal{P}_{\text{down}}, \mathcal{P}_{\text{up}}, \mathcal{P}_{\text{side}}\}$ is a partition of $\mathcal{P}'$.

Notice that if $(x, y) \in \mathcal{P}_{\text{down}} \cup \mathcal{P}_{\text{side}}$, then $\mathbf{G}'_x = \mathbf{G}_x$, $\mathbf{G}'_y = \mathbf{G}_y$, therefore $\mathsf{potential}_{(\mathbf{G}',D')}(x, y) = \mathsf{potential}_{(\mathbf{G},D)}(x, y)$.

Suppose now that $(x, y) \in \mathcal{P}_{\text{up}}$. We apply Lemma 7.5.12 on $\mathbf{G}$ by setting $i_1 = x$, $i_2 = y$, $i_3 = a$, and $\mathbf{G}_{\text{new}} = \mathbf{G}_b$ and deduce that $\mathbf{G}'_x \equiv_{\mathbf{p},t} \mathbf{G}'_y$ and $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}'_x, \mathbf{G}'_y) = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_x, \mathbf{G}_y)$. This implies that

$$\mathsf{potential}_{(\mathbf{G}',D')}(x, y) = \mathsf{potential}_{(\mathbf{G},D)}(x, y).$$

Thus, as the potentials of the pairs in $\mathcal{P}'$ do not change, we conclude that:

$$\begin{aligned} \mathsf{potential}_{(\mathbf{G}',D')}(\mathcal{P}') &= \mathsf{potential}_{(\mathbf{G},D)}(\mathcal{P}) - \mathsf{potential}_{(\mathbf{G},D)}(a, b) \\ &= k - \ell \end{aligned}$$

□

The induction hypothesis, along with the claim above, shows that:

$$\mathbf{p}(G') \;\geq\; k - \ell \tag{7.34}$$

Recall now that $\mathbf{G}_a \equiv_{\mathbf{p},t} \mathbf{G}_b$ and $\mathbf{G}'_a = \mathbf{G}_b$. This implies that $\mathbf{G}_a \equiv_{\mathbf{p},t} \mathbf{G}'_a$ and that $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}_a, \mathbf{G}'_a) = \ell$. We have:

$$\forall \mathbf{F} \in \mathcal{B}^{(t)}, \quad \mathbf{p}(\mathbf{G}_a \oplus \mathbf{F}) - \mathbf{p}(\mathbf{G}'_a \oplus \mathbf{F}) = \ell \tag{7.35}$$

By applying (7.35) for $\mathbf{F} = \overline{\mathbf{G}}_a$ and taking into account that $\mathbf{G}_a \oplus \overline{\mathbf{G}}_a = G$ and $\mathbf{G}'_a \oplus \overline{\mathbf{G}}_a = G'$, we conclude that $\mathbf{p}(G') = \mathbf{p}(G) - \ell$. This, together with (7.34), implies that $k \leq \mathbf{p}(G)$ as required. □

LEMMA 7.6.13. Let $\mathbf{p}$ be a graph parameter that has FII and is minor-closed and let $t \in \mathbb{N}$. If $\mathbf{G}$ is a boundaried graph, D is a binary rooted tree-decomposition of $\mathbf{G}$, and (a, b) is a transition pair of $(\mathbf{G}, D)$, then $\mathsf{potential}_{(\mathbf{G},D)}(a, b) \geq 0$.

Proof. Let $\mathbf{Q} = \mathsf{rep}_{\mathbf{p}}(\mathbf{G}_a) = \mathsf{rep}_{\mathbf{p}}(\mathbf{G}_b)$. Recall that $\mathsf{potential}_{(\mathbf{G},D)}(a, b) = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_a, \mathbf{G}_b)$. It remains to prove that $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}_a, \mathbf{G}_b) \geq 0$.

Let $t' = t_a = t_b$ and $\mathbf{H} = (H_a, \chi(a), \lambda_a)$. Notice that $\mathbf{G}_a \oplus \mathbf{H} = G_a$ and $\mathbf{G}_b \oplus \mathbf{H} = G_b$. Recall that $\mathbf{G}_a \equiv_{\mathbf{p},t} \mathbf{G}_b$ implies that:

$$\forall \mathbf{F} \in \mathcal{B}^{(t')}, \quad \mathbf{p}(\mathbf{G}_a \oplus \mathbf{F}) - \mathbf{p}(\mathbf{G}_b \oplus \mathbf{F}) = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_a, \mathbf{G}_b) \tag{7.36}$$

By applying (7.36) for $\mathbf{F} = \mathbf{H}$, we have that

$$\mathsf{transp}_{\mathbf{p}}(\mathbf{G}_a, \mathbf{G}_b) = \mathbf{p}(G_a) - \mathbf{p}(G_b).$$

Recall that $G_b \leq_{\mathrm{m}} G_a$. As $\mathbf{p}$ is minor closed, it holds that $\mathbf{p}(G_b) \leq \mathbf{p}(G_a)$ and the lemma follows. □

We now prove a combinatorial lemma asserting that, if T is the tree in a binary tree decomposition of G such that $|T| > c \cdot \mathbf{p}(G)$, for some constant

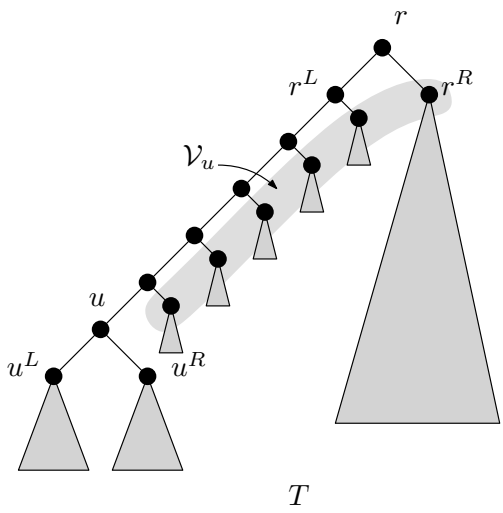


Figure 7.11: The set $\mathcal{V}_u$, the sub-trees and the vertices defined in the proof of Lemma 7.6.14, where $u = f(T, r)$.

$c \geq 1$, then T contains a pair collection of size bigger than $\mathbf{p}(G)$ and capacity "big enough" so as to guarantee the existence of a pair collection of more than $\mathbf{p}(G)$ transition pairs. As we prove, such a collection $\mathcal{C}$ can be constructed in $O(n)$ steps.

Lemma 7.6.14. There exists an algorithm that, takes as input a rooted binary tree (T, r) and an integer $z \geq 2$, and outputs – in $O_z(|T|)$ steps – a pair collection that has minimum capacity at least z and maximum capacity at most $2^z - 1$, and numbers more than

$$\frac{|T| + 1}{(z-1)^2 + z}$$

elements.

Proof. In our proof we "arrange" T so that for every two leafs of T the one in the left is the one of the bigest depth.

Let f be a function that takes as input a rooted binary tree (T, r) and finds a vertex u such that:

- $\mathsf{capacity}_{(T,r)}(r, u) \geq z$,
- rTu is the leftmost such path in T, and
- $|V(rTu)|$ is the minimum possible,

or outputs null when there is no such vertex in T. Notice that if $f(T, r) = \mathsf{null}$ then $|T| \leq z - 1$. As we only need to check the leftmost part of the tree and the vertices that have distance at most z from r, f runs in $O_z(1)$ steps.

We denote the left child of a vertex u by u^L and the right by u^R. We also define the set

$$\mathcal{V}_u = \bigcup_{v \in V(rTu)} \mathsf{children}_{T,r}(v) \setminus (V(rTu) \cup \mathsf{children}_{T,r}(u))$$

(see Figure 7.11). Notice that, if u is the output of f, then $|\mathcal{V}_u| \leq z - 1$.

We define a pair collection $C(T, r)$ using the following recursive formula:

$$C(T,r) = \begin{cases} C(T_{f(T,r)^L}, f(T,r)^L) \cup & \text{, if } |T| > z \text{ and} \\ C(T_{f(T,r)^R}, f(T,r)^R) \cup & \bigwedge_{v \in \mathcal{V}_{f(T,r)}} |T_v| \leq z-1 \\ \{(r, f(T,r))\} & \\ C(T_{r^L}, r^L) \cup C(T_{r^R}, r^R) & \text{, if } |T| > z \text{ and} \\ & \bigvee_{v \in \mathcal{V}_{f(T,r)}} |T_v| \geq z \\ \{(r, f(T,r))\} & \text{, if } |T| = z \\ \emptyset & \text{, if } |T| < z \end{cases}$$

It is straightforward to show that the pairs in $C(T,r)$ are non-interfering and have capacity at least z. Let $a(T,r)$ be the number of elements of $C(T,r)$. This number can be computed from the following recurrence relation:

$$a(T,r) = \begin{cases} a(T_{f(T,r)^L}, f(T,r)^L) + & \text{, if } |T| > z \text{ and} \\ a(T_{f(T,r)^R}, f(T,r)^R) + & \bigwedge_{v \in \mathcal{V}_{f(T,r)}} |T_v| \leq z-1 \\ 1 & \\ a(T_{r^L}, r^L) + a(T_{r^R}, r^R) & \text{, if } |T| > z \text{ and} \\ & \bigvee_{v \in \mathcal{V}_{f(T,r)}} |T_v| \geq z \\ 1 & \text{, if } |T| = z \\ 0 & \text{, if } |T| < z \end{cases}$$

Claim 5. We claim that:

$$a(T,r) \geq \frac{|T|+1}{(z-1)^2+z}$$

Proof of Claim 5. The proof of this claim is by induction:

Clearly, if $|T| = z$, $C(T,r)$ contains only one pair, namely $(r, f(T,r))$, thus $a(T,r) = 1$. As $z + 1/((z-1)^2+z)$ for $z \geq 2$ is at most 1 the claim holds.

Let $|T| > z$ and let $u = f(T,r)$. We assume that the claim holds for every tree with at most $|T|-1$ vertices. We distinguish two cases:

Case 1: If $\bigwedge_{v \in \mathcal{V}_u} |T_v| \leq z-1$, then:

$$\begin{aligned} a(T,r) &= a(T_{u^L},u^L) + a(T_{u^R},u^R) + 1 \\ &\geq \frac{|T_{u^L}|+1}{(z-1)^2+z} + \frac{|T_{u^R}|+1}{(z-1)^2+z} + 1 \\ &\geq \frac{|T_{u^L}|+|T_{u^R}|+2}{(z-1)^2+z} + 1 \end{aligned}$$

Notice that $|T_{u^L}|+|T_{u^R}| \geq |T| - (|\mathcal{V}_u|\cdot(z-1)+|rTu|) \geq |T| - ((z-1)\cdot(z-1)+z)$ (see Figure 7.11). Therefore:

$$\begin{aligned} a(T,r) &\geq \frac{|T|-(z-1)^2-z+2}{(z-1)^2+z} + 1 \\ &= \frac{|T|+2}{(z-1)^2+z} \\ &> \frac{|T|+1}{(z-1)^2+z} \end{aligned}$$

Case 2: If $\bigvee_{v\in\mathcal{V}_{f(T,r)}} |T_v| \geq z$ then:

$$\begin{aligned} a(T,r) &= a(T_{r^L},r^L) + a(T_{r^R},r^R) \\ &\geq \frac{|T_{r^L}|+1}{(z-1)^2+z} + \frac{|T_{r^R}|+1}{(z-1)^2+z} \\ &= \frac{|T|-1+2}{(z-1)^2+z} \\ &= \frac{|T|+1}{(z-1)^2+z} \end{aligned}$$

Hence, the claim holds for every case. □

Notice that, for every pair $(x, y) \in C(T, r)$, as we always pick y such that xT_xy is the leftmost and $|(xT_xy)|$ the minimum possible, it follows that (x, y) has capacity at most $2^z - 1$. Also notice that the definition of $C(T, r)$ provides a dynamic programming algorithm that computes this collection in $O_z(|T|)$ steps. □

The second main component of the algorithm of Theorem 7.3.1 is the procedure in the following lemma. We have to stress that to keep the running time of this algorithm linear we have to repetitively apply a bottom-up "compression" on a rich protrusion, that simultaneously "scans", "revises", and "compresses" it, alongside with its tree decomposition. To do so, it uses the *Boundaried Graph Compression Algorithm* of Lemma 7.6.6. This enforces that we are only processing parts of the graph of constant size. This procedure takes time proportional to the size of the rich protrusion, i.e., $O(n)$ steps.

Lemma 7.6.15. Let $\mathbf{p}$ be a graph parameter that is computable, has FII, and is minor-closed. For every $t \in \mathbb{N}$, there is an algorithm with the following specifications:

	Protrusion Compression Algorithm
Input:	A graph G^* and a $(t, \delta_{\mathbf{p}})$-rich protrusion R of G^* for $\mathbf{p}$.
Output:	A graph G^{**} where: 1. $\|G^{**}\| < \|G^*\|$, 2. $G^{**} \leq_{\text{m}} G^*$, and 3. $\mathbf{p}(G^{**}) = \mathbf{p}(G^*)$.

This algorithm runs in $O_t(|G^*|)$ steps.

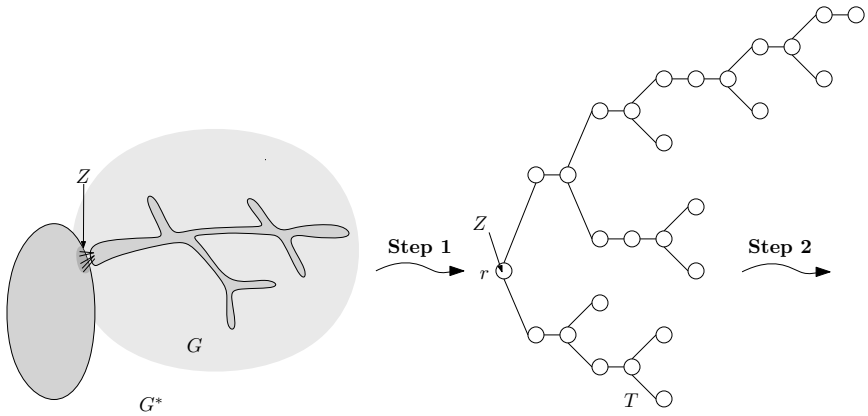


Figure 7.12: The first step of pre-processing of the algorithm of Lemma 7.6.15.

Proof. Recall that $\delta_{\mathbf{p}}(x) = x((4x \cdot \mu_{\mathbf{p}}(x) - 1)^2 + 4x \cdot \mu_{\mathbf{p}}(x))$. We set $Z = \partial_{G^*}(R)$, $G = G^*[R]$, and $\mathbf{G} = (G, Z, \lambda|_R)$ where λ is a some labelling of G. As R is a $(t, \delta_{\mathbf{p}})$-rich protrusion of G^* for $\mathbf{p}$, $|R| \geq \delta_{\mathbf{p}}(t) \cdot \mathbf{p}(G)$, thus $\mathbf{p}(G) = O(|R|) = O(|G|)$. We will do some pre-processing in G that consists of *3 steps*.

First the algorithm constructs a binary tree-decomposition of $\mathbf{G}$ – of width $t - 1$ – using the algorithm of Lemma 7.5.10 (see Figure 7.12, Step 1). This pre-processing takes $O_t(|G|)$ steps. Let $D = (T, \chi, r)$ be this binary tree-decomposition of $\mathbf{G}$.

Notice that $|T| \geq |R|/t$, because of Lemma 7.5.5. We use the algorithm of Lemma 7.6.14, in order to find in T a pair collection $\mathcal{Q}$ of (T, r) of mininum capacity $z = 4t \cdot \mu_{\mathbf{p}}(t)$ and maximum capacity $\alpha = 2^z - 1$ (see Figure 7.13, Step 2), where:

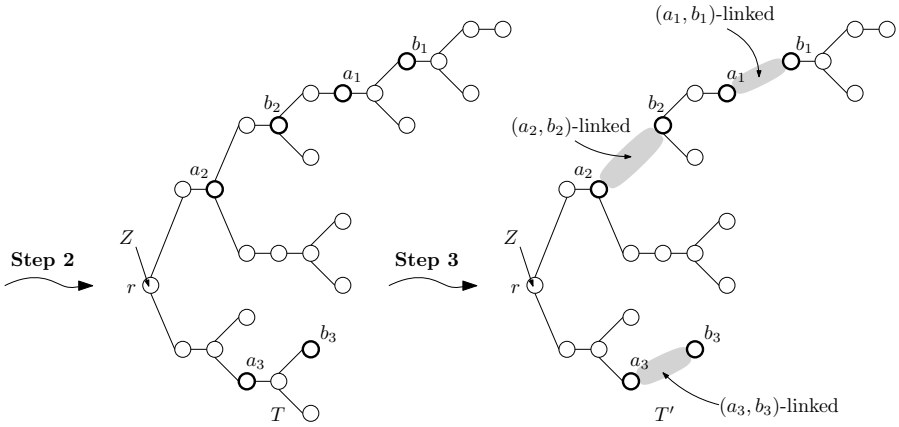


Figure 7.13: The last two steps of pre-processing of the algorithm of Lemma
7.6.15.

$$\begin{aligned}
|\mathcal{Q}| &\geq \frac{|T|+1}{(z-1)^2+z} > \frac{|T|}{(z-1)^2+z} \\
&\geq \frac{|R|}{t((z-1)^2+z)} \\
&= \frac{|R|}{t((4t\cdot\mu_{\mathbf{p}}(t)-1)^2+4t\cdot\mu_{\mathbf{p}}(t))} \\
&= \frac{|R|}{\delta_{\mathbf{p}}(t)} \geq \mathbf{p}(G)
\end{aligned}$$

We assume that $\mathcal{Q} = \{(a_1, b_1), \ldots, (a_s, b_s)\}$ where we know that $s \geq \mathbf{p}(G) + 1$.

For every $i \in [s]$, we run first the algorithm of Lemma 7.6.3 for every $(a_i, b_i) \in \mathcal{Q}$ and obtain a binary tree-decomposition $D' = (T', \chi', r)$ of $\mathbf{G}$ – of width at most $t-1$ – such that, for every $i \in [s]$, D' is (a_i, b_i)-linked

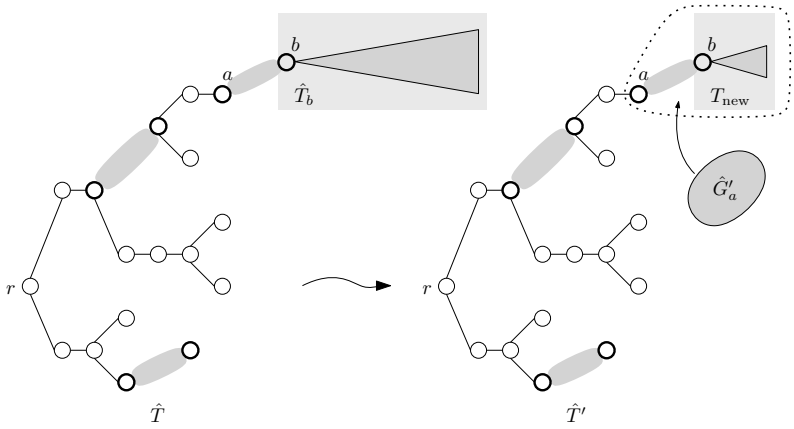


Figure 7.14: The main transformation of the algorithm of Lemma 7.6.15.

and $\mathsf{capacity}_{(T',r)}(a_i, b_i) \geq \frac{1}{4t} \cdot z \geq \mu_{\mathbf{p}}(t)$(see Figure 7.13, Step 3). As $\alpha = O_t(1)$, D' can be constructed in $O_t(|G|)$ steps.

From Lemma 7.6.4, we know that each $\mathsf{inner}_{T',r}(a_i, b_i), i \in [s]$ contains a vertical pair (a'_i, b'_i) that is a transition pair for $(\mathbf{G}, D')$. Let $\mathcal{Q}' = \{(a'_1, b'_1), \ldots, (a'_s, b'_s)\}$ be such a collection of transition pairs for $(\mathbf{G}, D')$ (here, we use Lemma 7.6.4 in order to prove the *existence* of $\mathcal{Q}'$ but *we do not construct it*). From Lemma 7.6.12, $|\mathcal{Q}'| = s > \mathbf{p}(G) \geq \mathsf{potential}_{(\mathbf{G},D')}(\mathcal{Q}')$, therefore $|\mathcal{Q}'| > \mathsf{potential}_{(\mathbf{G},D')}(\mathcal{Q}')$. From Lemma 7.6.13, all transition pairs in $\mathcal{Q}'$ are of non-negative potential, therefore $\mathcal{Q}'$ contains at least one null-transition pair, say (a, b). Clearly (a, b) is a vertical pair of T' where a, b belong in the same $\mathsf{inner}_{T',r}(a_i, b_i)$ for some $i \in [s]$. What remains is to find such a null-transition pair (a, b) in $O_t(|G|)$ steps. We do this by applying on $(\mathbf{G}, D')$ the following procedure, that makes use of the algorithm of Lemma 7.6.6:

Step 1: Set $\mu = \mu_{\mathbf{p}}(t)$, $\hat{\mathbf{G}} = \mathbf{G}$, $\hat{D} = D'$, and $\hat{\mathcal{Q}} = \mathcal{Q}$. We denote $\hat{D} = (\hat{T}, \hat{\chi}, r)$.

Step 2: Let $(a, b) \in \hat{\mathcal{Q}}$ such that b has maximum height in $(\hat{T}, r)$. Run the algorithm of Lemma 7.6.6 with input $\mathbf{G}_b$ and D_b and let $\mathbf{G}_{\text{new}}$ be the

output. Notice that $\mathbf{G}_{\text{new}}$ and $\mathbf{G}_b$ are strongly compatible, as in the algorithm of Lemma 7.6.6 the vertices of $\mathbf{G}_{\text{new}}$ originate from (equally labeled) vertices of $\mathbf{G}_b$. The algorithm considers the graph $\hat{\mathbf{G}}'$ that is constructed by replacing $\mathbf{G}_b$ by $\mathbf{G}_{\text{new}}$ in $\hat{\mathbf{G}}$ and as $\mathbf{tw}(\mathbf{G}_{\text{new}}) \leq t-1$ it builds a binary tree-decomposition $D_{\text{new}} = (T_{\text{new}}, \chi_{\text{new}}, b)$ and combines it with $\hat{D}^- = (\hat{T} \setminus \mathsf{desc}_{\hat{T},r}(b), \hat{\chi} \setminus \mathsf{desc}_{\hat{T},r}(b), b)$ to create a binary tree-decomposition $\hat{D}' = (\hat{T}', \hat{\chi}', r)$ of $\hat{\mathbf{G}}'$ (see Figure 7.14). Notice that $\hat{\mathcal{Q}}$ is a pair collection of $\hat{\mathbf{G}}'$ as well. Moreover, if for some $j \in [s]$, (a_j, b_j) is a null-transition pair in $\hat{\mathbf{G}}$, then it is also a null-transition pair in $\hat{\mathbf{G}}'$ as well. This follows if we apply Lemma 7.5.12 by setting $\mathbf{G} = \hat{G}$, $i_1 = a_j$, $i_2 = b_j$, $i_3 = b$, and $\mathbf{G}_{\text{new}} = \mathbf{G}_{\text{new}}$. We now consider all pairs (x, y) where $x, y \in \mathsf{inner}_{\hat{T}',r}(a, b)$ and check, with the help of Lemma 7.5.4 whether one, say (x, y), of them is a null-transition pair (see Figure 7.14). If this is the case, return (x, y) and **stop**. Clearly, as the capacity of (a, b) is $O_t(1)$ and as $|\mathbf{G}_{\text{new}}| = O_t(1)$ and using Lemma 7.5.5, we conclude that $|\hat{G}'_a| = O_t(1)$. Therefore the algorithm checks whether there is a null-transition pair in $\mathsf{inner}_{\hat{T}',r}(a, b)$ in $O_t(1)$ steps.

Step 3: If the previous step does not output a null-transition pair, then set $\hat{\mathbf{G}} = \hat{\mathbf{G}}'$, $\hat{D} = \hat{D}'$, and $\hat{\mathcal{Q}} = \hat{\mathcal{Q}} \setminus \{(a, b)\}$, and go to **Step 2**.

The algorithm above applies a bottom-up "compression" on $\mathbf{G}$ and D' and whenever it detects some pair in $\mathcal{Q}$ it checks – in $O_t(1)$ steps – whether its inner set contains some null-transition pair. As we have a guarantee that such a pair exists in $(\mathbf{G}, D')$ the algorithm will finally find it. The total number of compressions is proportional to the part of T' that does not belong in the inner set of some pair in $\mathcal{Q}$. Therefore, the above procedure runs in $O_t(|G|)$ steps, as required.

Let now (x, y) be a null-transition pair for $(\mathbf{G}, D')$ and let $(\mathbf{G}', D'')$ be the (x, y)-compression of $(\mathbf{G}, D')$. From Lemma 7.6.1 we deduce that $\mathbf{G}' \leq_{\text{m}} \mathbf{G}$, $|\mathbf{G}'| < |\mathbf{G}|$, $\mathbf{G}' \equiv_{\mathbf{p},t} \mathbf{G}$ and $\mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}') = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}_x, \mathbf{G}_y) = 0$.

Let $F = G^* \setminus \tilde{V}_r$, i.e., F contains all vertices of G^* except from the non-boundary vertices of $\mathbf{G}$. We set $\mathbf{F} = (G^*[F], \chi'(r), \lambda|_F)$ and observe that $G^* = \mathbf{F} \oplus \mathbf{G}$. Let $G^{**} = \mathbf{F} \oplus \mathbf{G}'$. From $|\mathbf{G}'| < |\mathbf{G}|$ and $\mathbf{G}' \leq_{\mathrm{m}} \mathbf{G}$, we obtain $|G^{**}| < |G^*|$, $G^{**} \leq_{\mathrm{m}} G^*$. Moreover, $\mathbf{G}' \equiv_{\mathbf{p},t} \mathbf{G}$ implies that:

$$\forall \mathbf{B} \in \mathcal{B}^{(t_r)}, \quad \mathbf{p}(\mathbf{G} \oplus \mathbf{B}) - \mathbf{p}(\mathbf{G}' \oplus \mathbf{B}) = \mathsf{transp}_{\mathbf{p}}(\mathbf{G}, \mathbf{G}') \tag{7.37}$$

By setting $\mathbf{B} = \mathbf{F}$ in (7.37), we obtain that $\mathbf{p}(G^*) - \mathbf{p}(G^{**}) = 0$ and the lemma follows. □

The proof of Theorem 7.3.1

We conclude that after detecting a rich protrusion in G, we can detect in $O(n)$ steps a null transition pair (a, b) and after applying a (a, b)-compression to it we can "compress" G to a new graph G' that is a proper minor of G and has $\mathbf{p}(G') = \mathbf{p}(G)$. As this compression cannot be applied for more than n steps, we finally construct an equivalent instance in $O(n^{2c+2})$ steps.

Proof of Theorem 7.3.1. The algorithm is the following:

Step 1: Apply the algorithm of Lemma 7.6.9 with input G, $\beta = 2 \cdot \mathsf{dec}(\mathbf{p})$ and $f = \delta_{\mathbf{p}}$. If this algorithm outputs that G does not contain any (β, f)-pair for $\mathbf{p}$, then set $G' = G$, output G', and **stop**. In this case, from Lemma 7.6.11, $|G| \leq c_{\mathbf{p}} \cdot \mathbf{p}(G)$ (recall that $c_{\mathbf{p}} = (\delta_{\mathbf{p}}(2 \cdot \mathsf{dec}(\mathbf{p})) \cdot (2 \cdot \mathsf{dec}(\mathbf{p}) \cdot \xi_{\mathbf{p}}(2 \cdot \mathsf{dec}(\mathbf{p})) + 1) + \mathsf{dec}(\mathbf{p}))$).

Step 2: If the algorithm of the previous step outputs a (β, f)-rich protrusion W of G for $\mathbf{p}$, then apply the algorithm of Lemma 7.6.15 for G and W and output a graph G' where $G' \leq_{\mathrm{m}} G$, $|G'| < |G|$, and $\mathbf{p}(G') = \mathbf{p}(G)$.

Step 3: Set $G = G'$ and go to **Step 1**.

Notice that the algorithm above outputs a graph G' that is a minor of the initial graph G.

Also $\mathbf{p}(G') = \mathbf{p}(G)$ and $|G'| \leq c_{\mathbf{p}} \cdot (\mathbf{p}(G')) \leq c_{\mathbf{p}} \cdot (\mathbf{p}(G))$, as $\mathbf{p}$ is minor-closed. As steps **1** and **2** cannot be applied more than $|G|$ times, step **1** takes $O_{c_{\mathbf{p}}}(|G|^{2\cdot \mathsf{dec}(\mathbf{p})+1})$ steps (using Lemma 7.6.9), and step **2** takes $O_{\beta}(|G|)$ steps (Lemma 7.6.15), the total running time is $O_{c_{\mathbf{p}}}(|G|^{2\cdot \mathsf{dec}(\mathbf{p})+2})$. □

7.7 Constructibility

We wish to make some short comments on the constructibility of the results presented in this Chapter. In general, the FII property does not imply that the algorithms in Proposition 7.4.1, as well as in 7.3.1 and Theorems 7.4.1, can be constructed. The same also holds for the constants of Theorem 7.4.2.

To construct the algorithm one needs to be able to construct a set of representatives for the equivalence classes of the relation $\equiv_{\mathbf{p},t}$ or – at least – of some refinement of it. This is typically possible when the problem in question can be expressed in CMSO logic or when some dynamic programming algorithm can solve the problem in graphs of bounded treewidth. For instance, this is possible for several families of problems including $\mathcal{F}$-Covering and $\mathcal{F}$-Packing.

7.7.1 An exercise on Computability

Before we wrap this Chapter up, we will show that it is not necessary to be given a representative collection for the algorithm of Theorem 7.3.1 to work. We can instead compute such a collection, provided that $\mathsf{card}_{\mathbf{p}}$ is a computable function.

LEMMA 7.7.1. Let $\mathbf{p}$ be a graph parameter. If $\mathbf{p}$ is computable, has FII, and $\mathsf{card}_{\mathbf{p}}$ is computable, then there exists an algorithm that given a $t \in \mathbb{N}$, outputs a *t-representative collection* for $\equiv_{\mathbf{p},t}$.

Proof. Let $k \geq t$. We define the class of boundaried graphs

$$\mathcal{G}_k^{(t)} = \{(G, X, \lambda) \in \mathcal{B}^{(t)} \mid |G| \leq k \text{ and } \lambda : V(G) \to [k]\}.$$

We also define the equivalence relation $\equiv^k_{\mathbf{p},t}$, where $\mathbf{G}_1 \equiv^k_{\mathbf{p},t} \mathbf{G}_2$ if and only if $\mathbf{G}_1, \mathbf{G}_2 \in \mathcal{G}_k^{(t)}$, are compatible, and

$$\exists c \in \mathbb{Z} \ \ \forall \mathbf{F} \in \mathcal{G}_k^{(t)}, \quad \mathbf{p}(\mathbf{G}_1 \oplus \mathbf{F}) = \mathbf{p}(\mathbf{G}_2 \oplus \mathbf{F}) + c$$

We will prove that for $\mathbf{G}_1, \mathbf{G}_2 \in \mathcal{G}_k^{(t)}$, if $\mathbf{G}_1 \not\equiv^k_{\mathbf{p},t} \mathbf{G}_2$, then $\mathbf{G}_1 \not\equiv_{\mathbf{p},t} \mathbf{G}_2$. From the fact that $\mathbf{G}_1 \not\equiv^k_{\mathbf{p},t} \mathbf{G}_2$ and the definition of $\equiv^k_{\mathbf{p},t}$ we conclude that for every $c \in \mathbb{Z}$ there exists $\mathbf{F} \in \mathcal{G}_k^{(t)}$ such that $\mathbf{p}(\mathbf{G}_1 \oplus \mathbf{F}) \neq \mathbf{p}(\mathbf{G}_2 \oplus \mathbf{F}) + c$. As $\mathcal{G}_k^{(t)} \subseteq \mathcal{B}^{(t)}$, it holds that

$$\forall c \in \mathbb{Z} \ \ \exists \mathbf{F} \in \mathcal{B}^{(t)}, \quad \mathbf{p}(\mathbf{G}_1 \oplus \mathbf{F}) \neq \mathbf{p}(\mathbf{G}_2 \oplus \mathbf{F}) + c$$

Therefore $\mathbf{G}_1 \not\equiv_{\mathbf{p},t} \mathbf{G}_2$.

Also notice that, for two boundaried graphs $\mathbf{G}_1, \mathbf{G}_2 \in \mathcal{G}_k^{(t)}$, if there exist $\mathbf{F}, \mathbf{F}' \in \mathcal{G}_k^{(t)}$ such that:

$$\mathbf{p}(\mathbf{G}_1 \oplus \mathbf{F}) - \mathbf{p}(\mathbf{G}_2 \oplus \mathbf{F}) \neq \mathbf{p}(\mathbf{G}_1 \oplus \mathbf{F}') - \mathbf{p}(\mathbf{G}_2 \oplus \mathbf{F}')$$

then $\mathbf{G}_1 \not\equiv^k_{\mathbf{p},t} \mathbf{G}_2$. To prove this, assume that $\mathbf{G}_1 \equiv^k_{\mathbf{p},t} \mathbf{G}_2$. From the definitions of $\equiv^k_{\mathbf{p},t}$ there exists a $c \in \mathbb{Z}$ such that

$$\begin{aligned} \mathbf{p}(\mathbf{G}_1 \oplus \mathbf{F}) &= \mathbf{p}(\mathbf{G}_2 \oplus \mathbf{F}) + c, \text{ and} \\ \mathbf{p}(\mathbf{G}_1 \oplus \mathbf{F}') &= \mathbf{p}(\mathbf{G}_2 \oplus \mathbf{F}') + c. \end{aligned}$$

Therefore

$$\mathbf{p}(\mathbf{G}_1 \oplus \mathbf{F}) - \mathbf{p}(\mathbf{G}_2 \oplus \mathbf{F}) = \mathbf{p}(\mathbf{G}_1 \oplus \mathbf{F}') - \mathbf{p}(\mathbf{G}_2 \oplus \mathbf{F}') = c,$$

a contradiction.

The algorithm that computes a representative collection $\mathcal{R}$ for $\equiv_{\mathbf{p},t}$, is the following:

Step 1: Compute $\mathsf{card}_{\mathbf{p}}(t)$. Let $\ell = \mathsf{card}_{\mathbf{p}}(t)$ and set $k = t$.

Step 2: Check for every $\mathcal{S} \subseteq \mathcal{G}_k^{(t)}$ with $|\mathcal{S}| = \ell$ whether for every two boundaried graphs $\mathbf{G}_i, \mathbf{G}_j \in \mathcal{S}$ there exist $\mathbf{F}, \mathbf{F}' \in \mathcal{G}_k^{(t)}$ such that

$$\mathbf{p}(\mathbf{G}_i \oplus \mathbf{F}) - \mathbf{p}(\mathbf{G}_j \oplus \mathbf{F}) \neq \mathbf{p}(\mathbf{G}_i \oplus \mathbf{F}') - \mathbf{p}(\mathbf{G}_j \oplus \mathbf{F}').$$

If such a set, say $\mathcal{R}$, exists, return it and **stop**, otherwise set $k = k + 1$ and go to **Step 2**.

Assume that this algorithm stops and that $\mathcal{R}$ is its output. Then $\mathcal{R}$ contains $\mathsf{card}_{\mathbf{p}}(t)$ boundaried graphs that are not equivalent with respect of $\equiv^k_{\mathbf{p},t}$, hence not equivalent with respect of $\equiv_{\mathbf{p},t}$. Thus $\mathcal{R}$ is a representative collection for $\equiv_{\mathbf{p},t}$.

The last thing missing is to prove that this algorithm will eventually stop. Since $\mathbf{p}$ has FII it holds that $\mathsf{card}_{\mathbf{p}}(t) \in \mathbb{N}$, therefore, for a representative collection $\mathcal{R}$ for $\equiv_{\mathbf{p},t}$ there exists a $k \geq t$ such that $\mathcal{R} \subseteq \mathcal{G}_k^{(t)}$. Thus, the algorithm will stop after it examines the set $\mathcal{G}_{k'}^{(t)}$, where $k' \geq k$ is such that for every two boundaried graphs $\mathbf{G}_i, \mathbf{G}_j \in \mathcal{R}$ there exist $\mathbf{F}, \mathbf{F}' \in \mathcal{G}_{k'}^{(t)}$ such that

$$\mathbf{p}(\mathbf{G}_i \oplus \mathbf{F}) - \mathbf{p}(\mathbf{G}_j \oplus \mathbf{F}) \neq \mathbf{p}(\mathbf{G}_i \oplus \mathbf{F}') - \mathbf{p}(\mathbf{G}_j \oplus \mathbf{F}').$$

Therefore, $\mathcal{R}$ will be computed in a finite number of steps. □

7.8 Conclusion

In this chapter we introduced the concept of parameter-invariant and minor-monotone kernels and saw how can these kernels be used to compute obstruction sets for a variety of parameters.

An interesting question is whether the above framework can be extended so as to yield approximation kernels (as they were recently defined in [181]) for more general problems.

Notice that the running time of our kernels (Theorem 7.4.1 and Theorem 7.3.1) is $O(|G|^{2c+2})$ steps where $c = \mathsf{dec}(\mathbf{p})$. We believe that with the use of the *randomized contraction technique* (introduced in [163]), it is possible to implement a faster guess of rich protrusions and drop this running time to $f(c)\cdot n^{O(1)}$ (similar ideas have recently been used in [178]).

An issue related to minor-monotone kernels is at which point polynomial kernelization for minor-closed problems implies – or is implied from – polynomial bounds to the corresponding obstructions. An important problem in this direction is to optimally bound the size of the graphs in $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathbf{p}_{p\text{-}\Pi}, k)$, when p-Π is the $\mathcal{F}$-Covering problem. According to [171, 172], each graph in this set has size $O(k^c)$, where c is a constant depending (non-constructively) to the class $\mathcal{F}$ (actually the result of [171, 172] does not even demand the graphs in $\mathcal{F}$ to be connected). A consequence of Theorem 7.4.1 is that all H-topological minor free graphs in $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathbf{p}_{p\text{-}\Pi}, k)$ are of size at most $c \cdot k$, where c depends (constructively) to the choice of $\mathcal{F}$. This motivates us to conjecture that the general bound can be dropped to $c \cdot k^{O(1)}$. This would come in contrast to the fact (proved in [171, 172]) that there are no (polynomial) kernels of this size for the $\mathcal{F}$-Covering problem, for certain instantiations of $\mathcal{F}$.

CHAPTER 8

GRAPH SEARCHING

Graph Searching is considered to be a flourishing branch of Graph Theory that has a wealth of applications. On top of this, it provides us with a descriptive way to define many combinatorial notions and problems. In a nutshell, "classic" Graph Searching or *Guaranteed Search* can be described as follows:

> *We find ourselves in a network of corridors, or tunnels, crossing one another (these crossings correspond to vertices of a graph and tunnels to its edges). This network is the "board" of a two player game. The first player controls an evading entity*[1] *hiding somewhere in this network. The second player has at his disposal a set of mobile agents or pursuers*[2]*. In each round of the game the two players move their "pieces" through this network. The goal of the first player is to evade capture. On the opposite side, the second player tries to guar-*

[1] You may think of this entity as an evader, a fugitive, an explorer or even a robber.
[2] Usually called searchers or – in the robber paradigm – cops.

antee this capture, deterministically, without any probabilistic assumptions.

The first formulation of Graph Searching was given by Parsons back in 1976 [116], based on an idea of R. Breisch published in a speleology magazine in 1967 [104]. The "speleotopological" problem described by Breisch in that paper had to do with an unfortunate explorer, lost in a network of caves, and a team of rescuers that were trying to locate him. The question asked was *"Is it possible to find the minimum number of rescuers needed to guarantee that the explorer will be found, deterministically and independently of his moves (even if he mistakenly make the worst possible move in every given time of the search)?"*.

Nikolai Petrov (Николай Петров) came across this problem (and some of Parsons' results) in 1982, doing research in a completely different field[3]. He gave a second definition of this problem based on his formalism and framework [117]. In 1989 Petr Golovach (Пётр Головач) proved that these two definitions are equivalent and proposed a third, which later became the standard [108, 109].

Graph searching has an enormous number of variations, most of whom are motivated by problems in practice, and can be defined by slightly changing the capabilities of the evading entity or its pursuers. These applications grabbed the attention from researchers following many different disciplines such as *Discrete Mathematics*, *Computer Science* and – even – *Artificial Intelligence*. It also provided us with a formal, mathematical way to describe the notions of *persecution*, *escape*, *threatening*, and – as we will see in Chapter 9 – *sense of direction*.

In the following Section we give the definition of Golovach, and then present some of the most important variations.

[3]Petrov studied graph searching as a natural restriction of differential games in Euclidian spaces like the *cossacks and the robber* game [118] and the *princess and the monster problem* [110].

8.1 Edge, Node and Mixed Search

We are given a graph G, without multiple edges and loops, where the game will be played. When the game begins (i.e., there are no searchers in the graph) player number one choses an edge and places the fugitive[4]. Then player number two places a number of his searchers on the vertices of G. In the classic scenario, both the fugitive and the searchers can only move to neighbour edges and vertices, that is, along a path connected their current position to their new position.

According to the capabilities of the fugitive and the way an edge is considered "searched" or "cleaned" we can define many different search games. To get a rough idea, we present some of the characteristics a fugitive may have:

- He may be *lazy or inert*: He is forced to move only when a searcher threatens him, that is, a searcher approaches his position. Contrarily, he may be *agile*: He moves on each and every round of the game.
- He may be *visible* by the searchers, or *invisible* to them.
- He may have *bounded speed*: He can traverse a bounded number of edges in each round. Or, he may have *unbounded speed*: He can move to every edge of the graph, provided of course that this edge is connected with his current position.

The fugitive is *captured* if at some point he resides on an edge e and one of the following capturing cases occurs.

A: *Both endpoints of* e *are occupied by a searcher*.

B: *A searcher slides along* e, i.e., a searcher is moved from one endpoint of the edge to the other endpoint.

[4]Unless otherwise stated, we adopt the fugitive vs. searchers scenario.

A *search strategy* on a graph G is a finite sequence $\mathcal{S}$ containing moves of the following types.

$\mathsf{p}(v)$: placing a new searcher on a vertex v,

$\mathsf{r}(v)$: deleting a searcher from a vertex v,

$\mathsf{s}(v,u)$: sliding a searcher lactated on vertex v along the edge $\{v,u\}$ and placing it on u.

We will start with a fugitive who is *agile* and *omniscient*, i.e., he moves at any time in the most favourable – for him – position and is *invisible*, thus, the searchers strategy can be given "in advance", as it does not depend on the moves of the fugitive during it, but only on the graph.

Given a search $\mathcal{S}$, we denote by $E(\mathcal{S},i)$ the set of edges that are clean after applying the first i steps of $\mathcal{S}$, where by "clean" we mean that the search strategy can guarantee that none of its edges will be occupied by the fugitive after the i-th step. More formally:

- we set $E(\mathcal{S},0)=\emptyset$ and
- in step $i>0$ we define $E(\mathcal{S},i)$ as follows: First consider the set Q_i containing all the edges in $E(\mathcal{S},i-1)$ plus the edges of $E^{(i)}$ the set of edges that are cleaned after the i-th move because of the application of cases **A**, **B** or both (this will give us three variations, which will be defined shortly). Notice that $E^{(i)}$ may be empty. In particular, it may be non-empty in case the i-move is a placement move, will always be empty in case the i-th move is a removal move and will surely be non-empty in case the i-th move is a sliding move. In the third case, the edge along which the sliding occurs is called *the sliding edge* of $E^{(i)}$. Then, the set $E(\mathcal{S},i)$ is defined as the set of all edges in Q_i minus those for which there is a path starting from them and finishing in an edge not in Q_i. This expresses the fact

that the agile and omniscient fugitive could use any of these paths in order to occupy again some of the edges in Q_i.

Definition 8.1.1. In case $E(\mathcal{S}, i) \subset Q_i$, we say that the i-th move is a *recontamination-move*.

Notice that in such a case we have that $E(\mathcal{S}, i-1) \not\subseteq E(\mathcal{S}, i)$. We will discuss recontamination extensively in the following Section. The object of a search game is to clear all edges.

Definition 8.1.2. We call a search $\mathcal{S}$ *complete* if at some step all edges of G are clean, i.e., $E(\mathcal{S}, i) = E(G)$ for some i.

Now we can define the three major variations of, the so called, *Fugitive search games*.

Definition 8.1.3 (Golovach, 1989 [108, 109])**.** When an edge can by cleaned only after the application of case **A**, then he have *Edge search* or *Graph sweeping*.

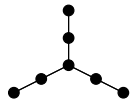
Definition 8.1.4 (Kirousis and Papadimitriou, 1985 [111, 112])**.** When an edge can by cleaned only after the application of case **B** and searchers cannot slide along an edge, then he have *Node search*.

Definition 8.1.5 (Bienstock and Robertson, 1991 [102])**.** When an edge can by cleaned after the application of both cases, then he have *Mixed search*[5].

According to the search game we are playing we call a search strategy $\mathcal{S}$ *edge search strategy, node search strategy* or *mixed search strategy*.

In Fugitive search games we are interesting to guarantee the capture of the fugitive, i.e., define a complete search strategy, using the smaller possible number of searchers. Therefore, we define the *cost* of a search strategy $\mathcal{S}$ on a graph G, denoted by $\mathsf{cost}_G(\mathcal{S})$, as the maximum number of searchers occupying vertices of G at the same time during $\mathcal{S}$.

[5] We should stress that in this variation searchers can slide along edges.



T

Figure 8.1: A graph with $\mathbf{es}(T) = \mathbf{ms}(T) = 2$ and $\mathbf{ns}(T) = 3$.

Definition 8.1.6. Let G be a graph. The *edge search number* of G is

$$\mathbf{es}(G) = \min\{\mathsf{cost}_G(\mathcal{S}) \mid \mathcal{S} \text{ is a complete edge search strategy for } G\}.$$

The *node search number* of G is

$$\mathbf{ns}(G) = \min\{\mathsf{cost}_G(\mathcal{S}) \mid \mathcal{S} \text{ is a complete node search strategy for } G\}.$$

The *mixed search number* of G is

$$\mathbf{ms}(G) = \min\{\mathsf{cost}_G(\mathcal{S}) \mid \mathcal{S} \text{ is a complete mixed search strategy for } G\}.$$

If $E(G) = \emptyset$ then $\mathbf{es}(G) = \mathbf{ns}(G) = \mathbf{ms}(G) = 0$.

Example 8.1.1. The tree T of Figure 8.1 has $\mathbf{es}(T) = \mathbf{ms}(T) = 2$ but $\mathbf{ns}(T) = 3$.

It is straight forward – from the definitions – that there exists some connection between these three search numbers. For instance, notice that mixed search unifies edge search and node search, as an edge search, or node search strategy $\mathcal{S}$ is also a mixed search strategy. Thus, for every graph G it holds that $\mathbf{ms}(G) \leq \mathbf{ns}(G)$ and $\mathbf{ms}(G) \leq \mathbf{es}(G)$. Moreover, one can use an extra searcher to simulate the sliding move along an edge to transform node search strategies to edge strategies and vice versa. All these facts are put together in the following observation.

Observation 8.1.1. For every graph G, the following are true:

(i) $\mathbf{ns}(G) - 1 \leq \mathbf{es}(G) \leq \mathbf{ns}(G) + 1$

(ii) $\mathbf{ms}(G) \leq \mathbf{es}(G) \leq \mathbf{ms}(G) + 1$

(iii) $\mathbf{ms}(G) \leq \mathbf{ns}(G) \leq \mathbf{ms}(G) + 1$

8.2 Monotonicity

Let us get back to the issue of *recontamination* in a search strategy. As the fugitive is agile and invisible, he can be everywhere in the graph. Let us think of him as a *poisonous gas* that has filled the whole graph. Then the purpose of the mobile agents is to clean this graph. They clean each edge, according to **A** or **B**, and when they occupy some vertex, they block the gas from entering in an edge having for endpoint this vertex. Of course, when they leave a vertex that is adjacent to uncleaned edges, they are not guarding it any more and – inevitably – a part of the graph that used to be clean now is recontaminated with the gas.

With what we have seen so far, as a search strategy that has some recontamination moves in it is still a search strategy, it must be clear that recontamination seems to be helpful for the searchers[6]! In this Section we plan to disprove it, but first we need to fix some notation and define three additional search numbers.

Definition 8.2.1. A search strategy $\mathcal{S}$ (edge, node or mixed) is called *monotone* if it does not contain any recontamination-moves.

Definition 8.2.2. Let G be a graph. The *monotone edge search number*

[6] We may be forced to clear multiple times the same edges but, by putting constrains in a search strategy we immediately get an upper bound in the search number. Remember, we are only interested in the number of searchers used and not in the time a search may take.

of G is

$$\mathbf{mes}(G) = \min\{\mathsf{cost}_G(\mathcal{S}) \quad | \quad \mathcal{S} \text{ is a complete monotone edge search strategy for } G\}.$$

The *monotone node search number* of G is

$$\mathbf{mns}(G) = \min\{\mathsf{cost}_G(\mathcal{S}) \quad | \quad \mathcal{S} \text{ is a complete monotone node search strategy for } G\}.$$

The *monotone mixed search number* of G is

$$\mathbf{mms}(G) = \min\{\mathsf{cost}_G(\mathcal{S}) \quad | \quad \mathcal{S} \text{ is a complete monotone mixed search strategy for } G\}.$$

If $E(G) = \emptyset$ then $\mathbf{mes}(G) = \mathbf{mns}(G) = \mathbf{mms}(G) = 0$.

The first proof of the *monotonicity* of a search game, i.e., a proof that for every non-monotone strategy $\mathcal{S}$ – with $\mathsf{cost}_G(\mathcal{S}) = k$ – there exists a monotone strategy $\mathcal{S}'$ with $\mathsf{cost}_G(\mathcal{S}') \leq k$, was given in 1983 by *Andrea LaPaugh*, but was publish ten years later.

Theorem 8.2.1 (Lapaugh, 1993 [113])**.** *For every graph G,* $\mathbf{es}(G) = \mathbf{mes}(G)$.

The second was given by Lefteris Kirousis (Λευτέρης Κυρούσης) and Christos Papadimitriou (Χρήστος Παπαδημητρίου) in 1986.

Theorem 8.2.2 (Kirousis and Papadimitriou, 1986 [112])**.** *For every graph G,* $\mathbf{ns}(G) = \mathbf{mns}(G)$.

The third one, which concerned the monotonicity of mixed search, was given by Daniel Bienstock and Paul Seymour in 1991, in the same paper they defined this game.

Theorem 8.2.3 (Bienstock and Seymour, 1991 [102])**.** *For every graph* G, $\mathbf{ms}(G) = \mathbf{mms}(G)$.

To sum up, monotonicity holds for the three basic games we defined. In 2003, Fedor V. Fomin (фёдор фомин) and Dimitrios M. Thilikos (Δημήτριος Θηλυκός) gave a min–max theorem that unified – and extend – all these monotonicity results [106]. Nevertheless, monotonicity does not hold in general. We will see an example in the next Section.

8.3 Connected Graph Searching

In many applications, searchers need to communicate with each other through a safe channel. Imagine for instance a network of computers infected by a virus. Sending messages to not infected computers will help a lot towards eliminating this virus, as you can coordinate your efforts. It is mandatory that the information exchanged will never reach an infected computer. In this example the "safe channel" is the part of the network that has already been cleaned.

Let us get back to graphs and the fugitive vs. searchers paradigm. Searchers need to communicate through the clean part of the graph and, in order for this communication to be safe – or, in other words, for the fugitive not being able to intersect it – the clean part of the graph must be connected. This concept of connectivity in graph searching was introduced in [100]. Let us make it more precise.

Definition 8.3.1. Let G be a graph and $\mathcal{S}$ a search strategy (edge, node or mixed). $\mathcal{S}$ is *connected* if and only if for every step i of $\mathcal{S}$, $E(\mathcal{S}, i)$ induces a connected subgraph of G.

We can – once more – revise the definitions of the three basic search games and allow only connected search strategies. In this way, we can define *connected edge search, connected node search*, and *connected mixed*

search, as well as their corresponding search numbers (denoted by **ces**, **cns** and **cms**). We have to stress that these numbers are also defined in disconnected graphs, but in this case their value is infinite.

We can even take it one step further and allow only connected and monotone strategies. We can then define *connected and monotone search games* and their search numbers (denoted by **cmes**, **cmns** and **cmms**). But, as was the case in the previous Section, if the connected variations of search games are monotone, by doing this we will not introduce some new parameters. Hence, a very interesting question emerges:

Does monotonicity hold for connected graph searching?

Other than the case of edge search we do not know the answer.

Theorem 8.3.1 (Yang, Dyer, and Alspach, [122])**.** *Let W be the graph of Figure 8.2, where every circle with the number i on the center, represents the clique of i vertices K_i, and the double lines between two cliques K_i and K_j represent a perfect matching between their vertices, if $i = j$, or between K_i and a subgraph of K_j, if $i < j$. It holds that* $\mathbf{ces}(W) = 281$ *but* $\mathbf{cmes}(W) = 290$.

This counter-example shows that, as far as connected edge search is concerned, searching a graph in a non-monotone way may be favourable for the searchers part. On the other hand, there are some graph classes where connected edge search is a monotone search game, trees for instance.

Theorem 8.3.2 (Barière, Flocchini, Fomin, Fraigniaud, Nisse, Santoro, and Thilikos [100])**.** *For every tree T,* $\mathbf{ces}(T) = \mathbf{cmes}(T)$.

Connected graph searching poses another difficulty. **ces**, **cns**, **cms** as well as **ces**, **cns**, **cmms** are not minor-closed parameters. This follows from the fact that deleting an edge form a graph may disconnect it and, thus, the graph obtained may have infinite connected search number.

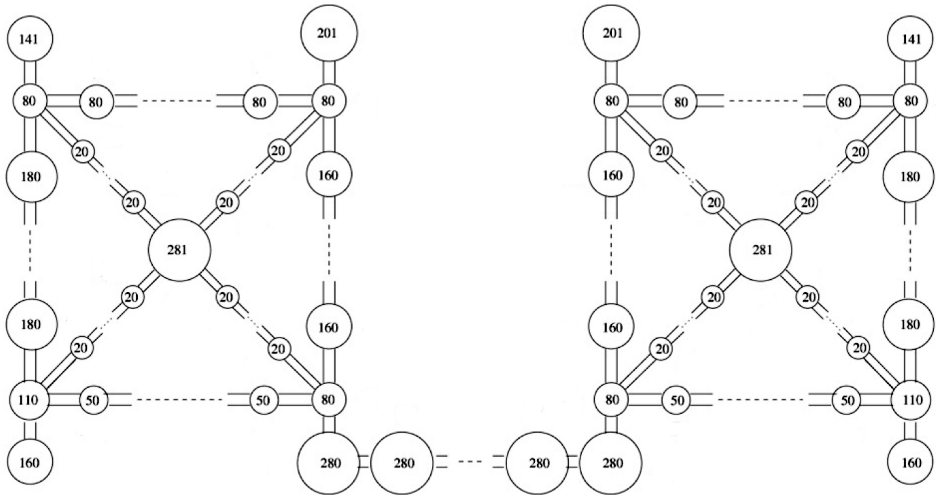


Figure 8.2: The graph W.

8.4 Width Parameters and Search Numbers

As we mentioned in Chapter 4, search numbers and width parameters are closely related. As a matter of fact, in most cases for each width parameter there exists an equivalent search number. To give you a rough idea why, think of a search strategy as an ordering of the edges being cleaned or the vertices visited by searchers. This can define an edge or vertex layout. Our goal is to minimize the number of searchers in the worst possible scenario, this leads to a min-max parameter, like the width parameter we defined.

Let us see some examples.

Theorem 8.4.1. *For every graph G,* $\mathbf{ns}(G) = \mathbf{pw}(G) + 1$.

The above follows by combining two results about *interval-thickness*, the first by Kirousis and Papadimitriou [111] and the second by *Rolf H. Möhring* [54].

THEOREM 8.4.2 (Thilikos, 2000 [120]). *For every graph G,* $\mathbf{lw}(G) \leq \mathbf{ms}(G) \leq \mathbf{lw}(G) + 1$.

The node search game variation where the fugitive is visible but lazy was first studied by Nick Dendris (Νικόλαος Δενδρής), Lefteris Kirousis, and Dimitrios Thilikos [105]. We denote by $\mathbf{ilns}(G)$ the minimum number of searchers that can guarantee the capture of a fugitive in this game, on the graph G. This search number can characterize the treewidth of a graph, providing us with a third definition of this parameter.

THEOREM 8.4.3 (Dendris, Kirousis, and Thilikos, 1997 [105]). *For every graph G,* $\mathbf{ilns}(G) = \mathbf{tw}(G) + 1$.

8.5 Obstructions Sets

Before moving any further, it would be useful to pointing out that:

THEOREM 8.5.1. *For every integer $k \geq 1$, the graph classes $\mathcal{G}[\mathbf{es}, k]$, $\mathcal{G}[\mathbf{ns}, k]$, and $\mathcal{G}[\mathbf{ms}, k]$ are closed under $\leq$, $\leq_{\text{in}}$, $\leq_{\text{c}}$ and $\leq_{\text{m}}$.*

This can be proven independently or by using the associated width parameters of these search numbers (and Observation 8.1.1). For connected search, the classes with bounded search number are only closed under takings of contractions, as the deletion of a vertex or an edge may "disconnect" the graph.

Finding obstruction sets for classes with bounded graph search number is a difficult endeavour. For small search numbers – really small, 1 and 2 for instance – some of these sets may be trivial. Bur then, thinks get very complicated.

EXAMPLE 8.5.1.

1. $\mathsf{obs}_{\leq_{\text{m}}}(\mathbf{ns}, 1)$ contains only the graph $(\{u, v\}, \{\{u, v\}\})$.

2. $\mathsf{obs}_{\leq_m}(\mathbf{ns}, 2) = \{K_3, T\}$, where T is shown in Figure 8.1.

3. $\mathsf{obs}_{\leq_m}(\mathbf{ns}, k)$, $\mathsf{obs}_{\leq_m}(\mathbf{cns}, k)$, and $\mathsf{obs}_{\leq_m}(\mathbf{cmns}, k)$, for $k = 1, 2$ are equal.

4. $\mathsf{obs}_{\leq_m}(\mathbf{es}, 1) = \mathsf{obs}_{\leq_m}(\mathbf{ces}, 1) = \mathsf{obs}_{\leq_m}(\mathbf{cmes}, 1) = \mathsf{obs}_{\leq_m}(\mathbf{ms}, 1) = \mathsf{obs}_{\leq_m}(\mathbf{cms}, 1) = \mathsf{obs}_{\leq_m}(\mathbf{cmms}, 1) = \{K_3, K_{1,3}\}$.

THEOREM 8.5.2 (Megiddo, Hakimi, Garey, Johnson, and Papadimitriou, [115])**.** *The obstruction set* $\mathsf{obs}_{\leq_m}(\mathbf{es}, 2)$ *consists of the 3 graphs shown in Figure 8.3.*

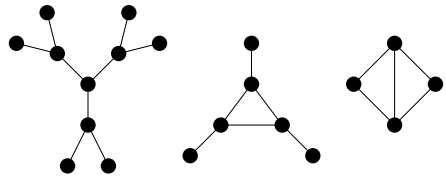


Figure 8.3: The set $\mathsf{obs}_{\leq_m}(\mathbf{es}, 2)$.

THEOREM 8.5.3 (Takahashi, Ueno, and Kajitani, [136])**.** *The obstruction set* $\mathsf{obs}_{\leq_m}(\mathbf{ms}, 2)$ *consists of the 36 graphs shown in Figure A.2.*

THEOREM 8.5.4 (Kinnersley and Langston, [133])**.** *The obstruction set* $\mathsf{obs}_{\leq_m}(\mathbf{ns}, 3)$ *consists of the 110 graphs shown in Figures A.3, A.4, A.5 and A.6.*

This is as far as our knowledge goes about obstruction sets for graph classes with bounded search numbers. In Chapter 9 we will make a small step forward and prove that $\mathsf{obs}_{\leq_c}(\mathbf{cms}, 2)$ and $\mathsf{obs}_{\leq_c}(\mathbf{cmms}, 2)$ are finite and that both consist of the same 177 graphs [1][7].

[7] We have to stress that these are contraction obstructions, thus there is no general proof that the obstruction sets would be finite

8.5.1 "Distance" to bounded search number

In this Section we will discuss the obstructions sets for the *distance to search number parameters*, i.e., $(\mathbf{ns}, r)$-dist, $(\mathbf{es}, r)$-dist and $(\mathbf{ms}, r)$-dist, for some $r \in \mathbb{N}$.

Observation 8.5.1. **ns**, **es** and **ms** are all big in grids.

Proof. Notice that, for every graph G, it holds that

$$\mathbf{tw}(G) \leq \mathbf{pw}(G) \leq \mathbf{ns}(G)$$

(this follows from the definition of **tw** and **pw**, and Theorem 8.4.1), and that

$$\mathbf{ns}(G) \leq \mathbf{es}(G) \leq \mathbf{ms}(G) + 1$$

(this follows from Observation 8.1.1).

Therefore, as $\mathbf{tw}(\boxplus_r) = r$ (Example 4.1.1), it holds that $\mathbf{ns}(\boxplus_r) \geq r$, $\mathbf{es}(\boxplus_r) \geq r$ and $\mathbf{ms}(\boxplus_r) \geq r - 1$. □

The following is an immediate corollary of Theorem 4.2.1 (using Theorem 8.5.1 and Observation 8.5.1).

Corollary 8.5.1. For every $k, r \in \mathbb{N}$, the graphs in $\mathsf{obs}_{\leq_\mathrm{m}}(\bullet, r)$, where $\bullet \in \{(\mathbf{ns}, r)\text{-dist}, (\mathbf{es}, r)\text{-dist}, (\mathbf{ms}, r)\text{-dist}\}$, have size bounded by k.

Since **ns**, **es** and **ms** are computable and – by definition – max parameters, the following is a direct corollary of Theorem 7.4.3 (again, using Theorem 8.5.1 and Observation 8.5.1).

Corollary 8.5.2. Let $r, k \in \mathbb{N}$ and let $\mathcal{C}$ be a graph class that is H-topological minor-free, for some graph H. Then the set $\mathsf{obs}_{\leq_\mathrm{m}}(\bullet, k) \cap \mathcal{C}$, where $\bullet \in \{(\mathbf{ns}, r)\text{-dist}, (\mathbf{es}, r)\text{-dist}, (\mathbf{ms}, r)\text{-dist}\}$, can be computed.

CHAPTER 9

CONNECTED GRAPH SEARCHING

In this chapter we are interested in obstruction characterizations for graphs with bounded connected (monotone) mixed search number. While we saw that **ms** is closed under taking of minors, this is *not* the case for **cms** and **cmms**, where the connectivity requirement applies. From Robertson – Seymour Theorem [84], the $\leq_{\mathrm{m}}$-obstruction set for the class of graphs with **ms** at most k is always finite. Recall that this set for $k = 1$ consists of 2 graphs, and for $k = 2$ of 36 graphs [119]. However, no such forbidden graph characterizations of the classes with bounded connected (monotone) mixed search number exists.

As we will see, **cms** and **cmms** are closed under takings of contractions, but, unfortunately, graphs are not well-quasi-ordered with respect to the contraction relation (see Figure 2.5), therefore, there is no guarantee that the contraction obstruction set for **cms** or **cmms** is finite for every positive integer $k \geq 2$ (the finiteness of this set is straightforward if $k = 1$ as $\mathsf{obs}_{\leq_{\mathrm{c}}}(\mathbf{cmms}, 1) = \{K_3, K_{1,3}\}$). In this Chapter we completely resolve the case where $k = 2$. We will show that $\mathsf{obs}_{\leq_{\mathrm{c}}}(\mathbf{cmms}, 2) = \mathsf{obs}_{\leq_{\mathrm{c}}}(\mathbf{cms}, 2)$ and prove that this set is finite by providing all 177 graphs

it contains. The proof is based on a series of lemmata that confine the structure of graphs with connected and monotone mixed search number at most 2.

We should stress that, in contrary to the case of **ms**, the *direction of searching* is *crucial* for **cms** and **cmms**. This makes the detection of the corresponding obstruction sets more elaborated, as special kind of obstructions are required in order to enforce a certain sense of direction in the search strategy. For this reason, the proof makes use of a more general variant of the mixed search strategy that forces the searchers to start from, and finish to, specific sets of vertices. Obstructions for this more general type of searching are combined in order to form the required obstructions for **cmms**.

At the end of this chapter we will present a double exponential lower bound on the size of the contraction obstruction set for the classes with bounded connected and monotone search number. Of course, this lower bound is only meaningful for the classes where this obstruction set is finite. If this is the case for some $k > 2$, this lower bound shows us that the size of $\mathsf{obs}_{\leq_c}(\mathbf{cmms}, k)$ will be huge.

9.1 Preliminary Definitions and Results

Instead of describing search strategies by explicitly stating each move of the searchers in any given round, it is much easier to describe it using an equivalent graph parameter. More often than not, this parameter can be defined using graph layouts. This is what we are going to do for connected (monotone) mixed search.

The following definition will be very useful.

Definition 9.1.1. Let A be a set and let $\mathcal{A} = \langle a_1, \ldots, a_r \rangle$ be an ordering of A. We denote by $\mathsf{prefsec}(\mathcal{A})$ the ordering $\langle A_0, \ldots, A_r \rangle$ of subsets of A, where $A_0 = \emptyset$ and for $i = 1, \ldots, r$, $A_i = \{a_1, \ldots, a_i\}$.

DEFINITION 9.1.2. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two disjoint orderings of A, we denote by $\mathcal{A}_1 \circ \mathcal{A}_2$ the *concatenation* of these two orderings.

In order to define an extension of the connected search game we have to define some new structures and fix some more notation.

DEFINITION 9.1.3. A *rooted graph triple*, or, for simplicity, a *rooted graph*, is an ordered triple $(G, S^{\text{in}}, S^{\text{out}})$ where G is a connected graph and S^{in} and S^{out} are subsets of $V(G)$ [1]. If $\mathbf{G} = (G, S^{\text{in}}, S^{\text{out}})$ then we also say that $\mathbf{G}$ is the graph G *in-rooted* at S^{in} and *out-rooted* at S^{out}.

DEFINITION 9.1.4. Given a rooted graph $\mathbf{G} = (G, S^{\text{in}}, S^{\text{out}})$, we define $\mathbf{rev}(\mathbf{G}) = (G, S^{\text{out}}, S^{\text{in}})$.

DEFINITION 9.1.5. Given a rooted graph $\mathbf{G} = (G, S^{\text{in}}, S^{\text{out}})$, where

$$S^{\text{in}} = \{v_1^{\text{in}}, \ldots, v_{|S^{\text{in}}|}^{\text{in}}\} \quad \text{and} \quad S^{\text{out}} = \{v_1^{\text{out}}, \ldots, v_{|S^{\text{out}}|}^{\text{out}}\},$$

we define its *enhancement* as the graph $\mathbf{enh}(G, S^{\text{in}}, S^{\text{out}})$ obtained from G after adding two vertices u^{in} and u^{out} and the edges in the sets

$$E^{\text{in}} = \{\{v_1^{\text{in}}, u^{\text{in}}\}, \ldots, \{v_{|S^{\text{in}}|}^{\text{in}}, u^{\text{in}}\}\}$$

and

$$E^{\text{out}} = \{\{v_1^{\text{out}}, u^{\text{out}}\}, \ldots, \{v_{|S^{\text{out}}|}^{\text{out}}, u^{\text{out}}\}\}$$

(See Figure 9.1).

From now on, we will refer to the vertices $u^{\text{in}}, u^{\text{out}}$ as the *vertex extensions* of $\mathbf{enh}(G, S^{\text{in}}, S^{\text{out}})$ and the edge sets E^{in} and E^{out} as the *edge extensions* of $\mathbf{enh}(G, S^{\text{in}}, S^{\text{out}})$.

[1] We stress that S^{in} and S^{out} are not necessarily disjoint sets.

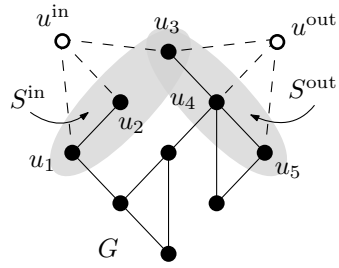


Figure 9.1: The enhancement of the rooted graph triple $(G, \{u_1, u_2, u_3\}, \{u_3, u_4, u_5\})$.

9.1.1 Connected search for rooted graphs

We have to stress that in fugitive search games we assume that searchers cannot make their first move in the graph before the fugitive makes his first move.

We define an extension of the connected search game.

Definition 9.1.6. Let G be a graph and let $S^{\text{in}}, S^{\text{out}} \subseteq V(G)$. A $(S^{\text{in}}, S^{\text{out}})$-*complete strategy for* G is a search strategy $\mathcal{S}$ on $\mathbf{enh}(G, S^{\text{in}}, S^{\text{out}})$ such that

(i) $E(\mathcal{S}, i) = E^{\text{in}}$, for some i,

(ii) $E(\mathcal{S}, i) \cap E^{\text{out}} = \emptyset$, for every i, and

(iii) $E(\mathcal{S}, i) = E(G) \setminus E^{\text{out}}$, for some i,

where $E^{\text{in}}, E^{\text{out}}$ are the edge extensions of $\mathbf{enh}(G, S^{\text{in}}, S^{\text{out}})$.

Based on the above definitions, we define $\mathbf{ms}(G, S^{\text{in}}, S^{\text{out}})$ as the minimum mixed search number over all possible $(S^{\text{in}}, S^{\text{out}})$-complete search strategies for G.

Similarly, we define **mms**$(G, S^{\text{in}}, S^{\text{out}})$ and **cmms**$(G, S^{\text{in}}, S^{\text{out}})$ where, in the case of connected searching, we additionally demand that S^{in} induces a connected subgraph of G. Notice that **ms**$(G) =$ **ms**$(G, \emptyset, \emptyset)$ and that this equality also holds for **mms** and **cmms**.

9.1.2 Expansions

Now we introduce a graph parameter equivalent to the search game defined in the previous paragraph. This parameter is defined using layouts of edge sets.

Definition 9.1.7. Given a graph G and a set $F \subseteq E(G)$, we define the *boundary of* F as follows:

$$\partial_G(F) = (\bigcup_{e \in F} e) \cap (\bigcup_{e \in E(G) \setminus F} e)$$

Definition 9.1.8. Let G be a graph and let E_1 and E_2 be subsets of $E(G)$. An (E_1, E_2)-*expansion* of G is an ordering $\mathcal{E} = \langle A_1, \ldots, A_r \rangle$ where:

(1) For $i \in \{1, \ldots, r-1\}$, $E_1 \subseteq A_i \subseteq E(G) \setminus E_2$.

(2) For $i \in \{1, \ldots, r-1\}$, $|A_{i+1} \setminus A_i| \leq 1$.

(3) $A_1 = E_1$.

(4) $A_r = E(G) \setminus E_2$.

Definition 9.1.9. Let G be a graph and let E_1 and E_2 be subsets of $E(G)$. An (E_1, E_2)-expansion of G is *connected* if, in addition to conditions (1) to (4), the following condition also holds:

(5) For $i \in \{1, \ldots, r\}$, $G[A_i]$ is connected.

Definition 9.1.10. Let G be a graph and let E_1 and E_2 be subsets of $E(G)$. An (E_1, E_2)-expansion of G is *monotone* if, in addition to conditions (1) to (4), the following condition also holds:

(6) $A_1 \subseteq \cdots \subseteq A_r$.

Definition 9.1.11. Let $i \in \{1, \ldots, r-1\}$. The *cost of an expansion $\mathcal{E}$ at position i* is defined as $\mathsf{cost}_G(\mathcal{E}, i) = |\partial_G(A_i)| + q_i$ where q_i is equal to 1 if one of the following holds:

- $|A_i| \geq 2$ and $A_i \setminus A_{i-1}$ contains a pendant edge of G
- A_i consists of only one edge that is an isolated edge of G.

If none of the above two conditions hold then q_i is equal to 0. The *cost of the expansion $\mathcal{E}$*, denoted as $\mathsf{cost}_G(\mathcal{E})$, is the maximum cost of $\mathcal{E}$ among all positions $i \in \{1, \ldots, r-1\}$.

Our parameter, denoted by $\mathbf{p}(G, S^{\text{in}}, S^{\text{out}})$ [2], is defined to be the minimum cost that an $(E^{\text{in}}, E^{\text{out}})$-expansion of $\mathbf{enh}(G, S^{\text{in}}, S^{\text{out}})$ may have, where $E^{\text{in}}, E^{\text{out}}$ are the edge extensions of $\mathbf{enh}(G, S^{\text{in}}, S^{\text{out}})$.

We also define $\mathbf{mp}(G, S^{\text{in}}, S^{\text{out}})$ (if we consider only monotone $(E^{\text{in}}, E^{\text{out}})$-expansions) and $\mathbf{cmp}(G, S^{\text{in}}, S^{\text{out}})$ (if we consider connected monotone $(E^{\text{in}}, E^{\text{out}})$-expansions). We finally define $\mathbf{cmp}(G) = \mathbf{cmp}(G, \emptyset, \emptyset)$.

Lemma 9.1.1. Let $(G, S^{\text{in}}, S^{\text{out}})$ be a rooted graph and let $S_1^{\text{in}} \subseteq S^{\text{in}}$ and $S_1^{\text{out}} \subseteq S^{\text{out}}$, where $G[S^{in}]$ is a connected subgraph of G. Then $\mathbf{cmp}(G, S_1^{\text{in}}, S_1^{\text{out}}) \leq \mathbf{cmp}(G, S^{\text{in}}, S^{\text{out}})$.

Proof. Let $E^{\text{in}}, E^{\text{out}}$ and $E_1^{\text{in}}, E_1^{\text{out}}$ be the edge extensions of $\mathbf{enh}(G, S^{\text{in}}, S^{\text{out}})$ and $\mathbf{enh}(G, S_1^{\text{in}}, S_1^{\text{out}})$ respectively. Notice that, as $S_1^{\text{in}} \subseteq S^{\text{in}}$ and $S_1^{\text{out}} \subseteq S^{\text{out}}$, $E_1^{\text{in}} \subseteq E^{\text{in}}$ and $E_1^{\text{out}} \subseteq E^{\text{out}}$.

[2] Due to lack of inspiration we do not give a distinct name to this parameter...

Let $\mathcal{E} = \langle A_1, \ldots, A_r \rangle$ be an monotone and connected $(E^{\text{in}}, E^{\text{out}})$-expansion of $\textbf{enh}(G, S^{\text{in}}, S^{\text{out}})$, with cost at most k.

As $G[S^{\text{in}}]$ is a connected subgraph of G, for every vertex of $S^{\text{in}} \setminus S_1^{\text{in}}$ there exists a path connecting it with a vertex of S_1^{in} that only uses vertices of S^{in}. We define the following edge sets:

- E_1^1 contains all edges that have a vertex of S_1^{in} and a vertex of $S^{\text{in}} \setminus S_1^{\text{in}}$ as endpoints. Let $V_1 = (\bigcup_{e \in E_1^1} e) \setminus S_1^{\text{in}}$, then E_1^2 contains all edges that have both endpoints in V_1.
- E_j^1 contains all edges that have a vertex of V_{j-1} and a vertex of $S = S^{\text{in}} \setminus (S_1^{\text{in}} \cup (\bigcup_{l=1\ldots,j-1} V_l))$ as endpoints. Let $V_j = (\bigcup_{e \in E_j^1} e) \setminus S$, then E_j^2 contains all edges that have both endpoints in V_j.

For each edge set E_j^i, $1 \leq j \leq d$ and $i \in \{1, 2\}$, where d is the maximum distance between a vertex of $S^{\text{in}} \setminus S_1^{\text{in}}$ to some vertex in S_1^{in}, we define arbitrarily an edge ordering L_j^i. We then define an ordering $\mathcal{E}_1$ of edge sets as follows:

- $A_1^{'} = (A_1 \setminus E^{\text{in}}) \cup E_1^{\text{in}}$
- $A_{1+l}^{'} = A_{1+l-1}^{'} \cup \hat{A}_l$ for $l = 1, \ldots, |E_1^1|$, where $\langle \hat{A}_1, \ldots, \hat{A}_{|E_1^1|} \rangle = \mathsf{prefsec}(L_1^1)$
- $A_{1+|E_1^1|+l}^{'} = A_{1+|E_1^1|+l-1}^{'} \cup \hat{A}_l$, for $l = 1, \ldots, |E_1^2|$, where $\langle \hat{A}_1, \ldots, \hat{A}_{|E_1^2|} \rangle = \mathsf{prefsec}(L_1^2)$
- $A_{1+|E_1^1|+|E_1^2|+\cdots+|E_{j-1}^1|+|E_{j-1}^2|+l}^{'} = A_{1+|E_1^1|+|E_1^2|+\cdots+|E_{j-1}^1|+|E_{j-1}^2|+l-1}^{'} \cup \hat{A}_l$, for $l = 1, \ldots, |E_j^1|$, where $\langle \hat{A}_1, \ldots, \hat{A}_{|E_j^1|} \rangle = \mathsf{prefsec}(L_j^1)$
- $A_{1+|E_1^1|+|E_1^2|+\cdots+|E_{j-1}^1|+|E_{j-1}^2|+|E_j^1|+l}^{'} = A_{1+|E_1^1|+|E_1^2|+\cdots+|E_{j-1}^1|+|E_{j-1}^2|+|E_j^1|+l-1}^{'} \cup \hat{A}_l$, for $l = 1, \ldots, |E_j^2|$, where $\langle \hat{A}_1, \ldots, \hat{A}_{|E_j^2|} \rangle = \mathsf{prefsec}(L_j^2)$

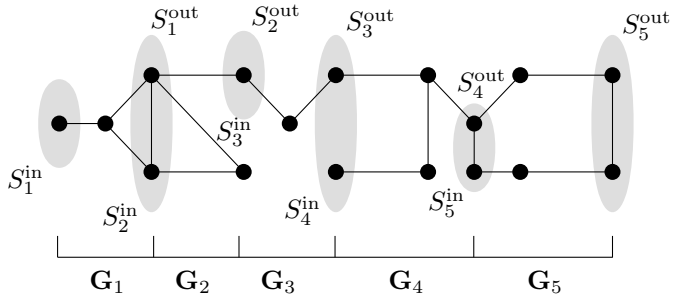


Figure 9.2: **glue**$(\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3, \mathbf{G}_4, \mathbf{G}_5)$.

Let $s = |E_1^1| + |E_1^2| + \cdots + |E_d^1| + |E_d^2|$. Notice that there exist a $l_0 \in \{2, \ldots, r\}$ such that $A'_{1+s} = (A_{l_0} \setminus E^{\text{in}}) \cup E_1^{\text{in}}$. We define a second ordering $\mathcal{E}_2$ of edge sets as follows: $A'_{1+s+l} = (A_{l_0+l} \setminus E^{\text{in}}) \cup E_1^{\text{in}}$ for $l = 1, \ldots, r - l_0$.

Clearly, $\mathcal{E}' = \mathcal{E}_1 \circ \mathcal{E}_2$ satisfies conditions (1) to (4) and therefore is an $(E_1^{\text{in}}, E_1^{\text{out}})$-expansion of **enh**$(G, S_1^{\text{in}}, S_1^{\text{out}})$. Moreover, the monotonicity and connectivity of $\mathcal{E}'$ follows from the monotonicity and connectivity of $\mathcal{E}$.

Notice that, for every $i \in \{1, \ldots, r\}$, $\partial_G(A_i) = \partial_G((A_i \setminus E^{\text{in}}) \cup E_1^{\text{in}})$ therefore $\mathsf{cost}_G(\mathcal{E}', i) \leq \mathsf{cost}_G(\mathcal{E}, i)$. From this we conclude that $\mathcal{E}'$ has cost at most k. □

We need an operation that will "glue together" rooted graphs by identifying their "in" and "out" roots.

Definition 9.1.12. Let $\mathbf{G}_1, \ldots, \mathbf{G}_r$ be rooted graphs such that $\mathbf{G}_i = (G_i, S_i^{\text{in}}, S_i^{\text{out}})$ where $V(G_i) \cap V(G_{i+1}) = S_i^{\text{out}} = S_{i+1}^{\text{in}}$, $i \in \{1, \ldots, r-1\}$. We define, **glue**$(\mathbf{G}_1, \ldots, \mathbf{G}_r) = (G_1 \cup \cdots \cup G_r, S_1^{\text{in}}, S_r^{\text{out}})$ (see Figure 9.2).

Lemma 9.1.2. Let $\mathbf{G}_1, \ldots, \mathbf{G}_r$ be rooted graphs such that $\mathbf{G}_i = (G_i, S_i^{\text{in}}$,$S_i^{\text{out}})$ where $V(G_i) \cap V(G_{i+1}) = S_i^{\text{out}} = S_{i+1}^{\text{in}}$, $i \in \{1, \ldots, r-1\}$. Then

$$\mathbf{cmp}(\mathbf{glue}(\mathbf{G}_1, \ldots, \mathbf{G}_r)) \leq \max\{\mathbf{cmp}(\mathbf{G}_i) \mid i \in \{1, \ldots, r\}\}.$$

Proof. Let $E_i^{\text{in}}, E_i^{\text{out}}$ be the edge extensions and $\mathcal{E}_i = \langle A_1^i, \ldots, A_{l_i}^i \rangle$ be an monotone and connected $(E_i^{\text{in}}, E_i^{\text{out}})$-expansion of $\mathbf{enh}(G_i, S_i^{\text{in}}, S_i^{\text{out}})$, for every $i \in \{1, \ldots, r\}$. Clearly $\mathcal{E} = \langle A_1^1, \ldots, A_{l_1}^1, A_{l_1}^1 \cup A_2^2, \ldots, A_{l_1}^1 \cup A_{l_2}^2, \ldots, (\cup_{1 \le i < r} A_{l_i}^i) \cup A_2^r, \ldots, (\cup_{1 \le i < r} A_{l_i}^i) \cup A_{l_r}^r \rangle$ is an $(E_1^{\text{in}}, E_r^{\text{out}})$-expansion of $\mathbf{glue}(\mathbf{G}_1, \ldots, \mathbf{G}_r)$ and, as expansions $\mathcal{E}_i$, $i \in \{1, \ldots, r\}$ are monotone and connected, conditions (5) and (6) hold.

We observe that $\mathsf{cost}_{G_1 \cup \cdots \cup G_r}(\mathcal{E}) \le \max\{\mathsf{cost}_{G_1}(\mathcal{E}_1), \ldots, \mathsf{cost}_{G_r}(\mathcal{E}_r)\}$, therefore $\mathbf{cmp}\ (\mathbf{glue}(\mathbf{G}_1, \ldots, \mathbf{G}_r)) \le \max\{\mathbf{cmp}(\mathbf{G}_i) \mid i \in \{1, \ldots, r\}\}$. □

Now we show the equivalence of our search game number to the parameter defined above.

LEMMA 9.1.3. For every graph G, $\mathbf{cmms}(G, S^{\text{in}}, S^{\text{out}}) = \mathbf{cmp}(G, S^{\text{in}}, S^{\text{out}})$.

Proof. Assume that $G^* = \mathbf{enh}(G, S^{\text{in}}, S^{\text{out}})$ has a complete search strategy $\mathcal{S}$ satisfying conditions (i) – (iii) with cost at most k. We construct an edge ordering of $E(G)$ as follows.

Observe that, because of the monotonicity of $\mathcal{S}$, $E^{(i)} = E(\mathcal{S}, i) \setminus E(\mathcal{S}, i-1)$. For every $i \in \{1, \ldots, |\mathcal{S}|\}$, we define L_i by taking any ordering of the set $E^{(i)}$ and insisting that, if $E^{(i)}$ contains some sliding edge, this edge will be the first edge of L_i. Let $\mathcal{E} = \langle A_0, \ldots, A_r \rangle$ be the sequence of prefixes of $L_1 \circ \ldots \circ L_{|S|}$, including the empty set (that is $A_0 = \emptyset$). Notice that, because of Condition (i), $A_s = E^{\text{in}}$ for some $s \in \{1, \ldots, |\mathcal{S}|\}$, and, because of Condition (iii), $A_t = E(G) \setminus E^{\text{out}}$, for some $t \in \{1, \ldots, |\mathcal{S}|\}$. We now claim that $\mathcal{E}' = \langle A_s, \ldots, A_t \rangle$ is an $(E^{\text{in}}, E^{\text{out}})$-expansion of G^*. Indeed, Condition (1) holds because of Condition (ii) and Conditions (2) – (4) hold because of the construction of $\mathcal{E}'$. Moreover, the connectivity and the monotonicity of $\mathcal{E}'$ follow directly from the connectivity and the monotonicity of $\mathcal{S}$.

It remains to prove that the cost of $\mathcal{E}'$ is at most k. For each $j \in \{0, \ldots, |\mathcal{E}'|\}$ we define i_j such that the unique edge in $A_j \setminus A_{j-1}$ is an edge in $E^{(i_j)}$ and we define h_j such that $A_{h_j} \setminus A_{h_j - 1}$ contains the fist

edge of L_{i_j}. Notice now that the cost of $\mathcal{E}'$ at positions h_j to j is upper bounded by the cost of $\mathcal{E}'$ at position h_j. Therefore, it is enough to prove that the cost of $\mathcal{E}'$ at position h_j is at most k. Recall that this cost is equal to $|\partial_G(A_{h_j})| + q_{h_j}$. We distinguish two cases:

Case 1. If $q_{h_j} = 0$, then the cost of $\mathcal{E}'$ at position h_j is equal to $|\partial_G(A_{h_j})|$. As $\mathcal{S}$ is monotone, all vertices in $\partial_G(A_{h_j})$ should be occupied by searchers after the i_j-th move of $\mathcal{S}$ and therefore the cost of $\mathcal{E}'$ at position h_j is at most k.

Case 2. If $q_{h_j} = 1$, then the i_j-th move of $\mathcal{S}$ is either the placement of a searcher on a pendant vertex x or the sliding of a searcher along a pendant edge $\{y, x\}$ towards its pendant vertex x. In both cases, $x \notin \partial_G(A_{h_j})$ and all vertices in $\partial_G(A_{h_j})$ should be occupied by searchers after the i_j-th move. In the first case, there are in total at least $|\partial_G(A_{h_j})| + 1$ searchers on the graph and we are done. In the second case, we observe that, due to monotonicity, $\partial_G(A_{h_j}) = \partial_G(A_{h_j-1}) \setminus \{y\}$. As after the $(h_j - 1)$-th move all vertices of $\partial_G(A_{h_j-1})$ were occupied by searchers, we obtain that $|\partial_G(A_{h_j})| \leq k - 1$ and thus the cost of $\mathcal{E}'$ at position h_j is at most k.

Now assume that there exist a monotone and connected $(E^{\text{in}}, E^{\text{out}})$-expansion of G^*, say $\mathcal{E} = \langle A_1, \ldots, A_r \rangle$, with cost at most k. We can additionally assume that $\mathcal{E}$ is properly monotone; this can be done by discarding additional repetitions of a set in $\mathcal{E}$.

Moreover, starting from $\mathcal{E}$, we can construct a monotone and connected $(E^{\text{in}}, E^{\text{out}})$-expansion of G^*, with cost at most k, say $\mathcal{E}' = \langle A'_1, \ldots, A'_r \rangle$, with the following additional property:

Expansion property: For every $i \in \{1, \ldots, r-1\}$ for which $V(A'_i) \subset V(A'_{i+1})$, A'_i contains all edges of G^* with both endpoints in $V(A'_i)$.

This can be accomplished by a series of appliances of the following

rule:

Rule: Let $V(A_i) \subset V(A_{i+1})$ for some i and let $L = \langle e_1, \ldots, e_n \rangle$ be an ordering of the edges $E(G^*) \setminus A_i$ with both endpoints in $V(A_i)$. For every $j \leq i$ define $A'_j = A_j$. Then, define $A'_{i+1} = A_i \cup \{e_1\}$, $A'_{i+2} = A_i \cup \{e_1, e_2\}$ and so on, until $A'_{i+n} = A_i \cup \{e_1, \ldots, e_n\}$. Finally, for every $j \geq i + n$, define $A'_j = A_j \cup \{e_1, \ldots, e_n\}$.

One can easily check that, after every application of this rule, the constructed sequence of edge sets is indeed an $(E^{\text{in}}, E^{\text{out}})$-expansion of G^* and, furthermore, it is monotone and connected. Notice that, for $j = 1, \ldots, n$, $\partial_{G^*}(A'_{i+j}) \subseteq \partial_{G^*}(A_i)$ and for $j \geq i+n$, $|\partial_{G^*}(A'_j)| \leq |\partial_{G^*}(A_j)|$. Moreover, if $|A_i| \geq 2$ and $A_i \setminus A_{i-1}$ contains a pendant edge of G^* then for every $j \in \{1, \ldots, n\}$, $|A'_{i+j}| \geq 2$ and $A'_{i+j} \setminus A'_{i+j-1}$ contains the same pendant edge of G^*, hence the cost of $\mathcal{E}'$ is at most k (notice that if A_i consists of only one edge that is an isolated edge of G^* then there does not exist an edge in $E(G^*) \setminus A_i$ with both endpoints in $V(A_i)$, therefore we do not need to apply this rule).

For the rest of the proof, we consider that the Expansion property holds for the given $(E^{\text{in}}, E^{\text{out}})$-expansion of G^*.

Our target is to define a $(S^{\text{in}}, S^{\text{out}})$-complete monotone search strategy $\mathcal{S}$ of G^* with cost at most k.

The first $|S^{\text{in}}|$ moves of $\mathcal{S}$ will be $\mathsf{p}(u^{\text{in}})$ and the next $|S^{\text{in}}|$ will be $\mathsf{s}(u^{\text{in}}, v_i^{\text{in}})$. We denote this sequence of moves by $\mathcal{S}_0$. Notice that $E(\mathcal{S}, 2|S^{\text{in}}|) = A_1$.

For every vertex u in the set $V^* = V(G^*) \setminus S^{\text{in}} \setminus \{u^{\text{out}}\}$, we define l_u to be the first integer in $\{1, \ldots, r\}$ such that $u \in V(A_{l_u})$.

Let $L = \langle u_1, \ldots, u_{|V^*|} \rangle$ be an ordering of V^* such that $i \leq j$ when $l_{u_i} \leq l_{u_j}$. Notice that, for each $i \in \{1, \ldots, |V^*|\}$, the vertex u_i is an endpoint of the unique edge e_i in $A_{l_{u_i}-1} \setminus A_{l_{u_i}}$ and let v_i be the other endpoint of e_i. Notice that, due to the connectivity and the monotonicity of $\mathcal{E}$, $v_i \in \partial_{G^*}(A_{l_{u_i}-1})$. We also observe that u_i is pendant if and only

if $u_i \notin \partial_{G^*}(A_{l_{u_i}})$. We define $E' = \{e_1, \ldots, e_{|V^*|}\}$ and we call a set A_j, $j \in \{1, \ldots, r\}$, *crucial* if and only if $|A_{j-1} \cap E'| < |A_j \cap E'|$.

For each $i \in \{1, \ldots, |V^*|\}$, we define a sequence $\mathcal{S}_i$ of moves as follows: If $v_i \in \partial_{G^*}(A_{l_{u_i}})$ then the first move of $\mathcal{S}_i$ is $\mathsf{p}(u_i)$, otherwise it is $\mathsf{s}(v_i, u_i)$. The rest of the moves in $\mathcal{S}_i$ are the removals – one by one – of the searchers in $\partial_{G^*}(A_{l_{u_i}-1}) \setminus \partial_{G^*}(A_{l_{u_i}})$. Then, we define $\mathcal{S} = \mathcal{S}_0 \circ \mathcal{S}_1 \circ \cdots \circ \mathcal{S}_{|V^*|}$.

Notice that, according to the expansion property, all edges of the sets A_j, for $j = 1, \ldots, l_{u_1}$ have both endpoints in S^{in}. Moreover, for every $i \in \{1, \ldots, |V^*| - 1\}$ all edges of the sets A_j, for $j = l_{u_i}, \ldots, l_{u_i+1} - 1$, have both endpoints in $V(A_{l_{u_i}})$ and all edges of the sets A_j, for $j = l_{u_{|V^*|}}, \ldots, r$, have both endpoints in $V(A_{l_{u_{|V^*|}}})$.

First we show that the following claim is true:

Claim 6. For every A_j, $j \in \{1, \ldots, r\}$, the vertices of $\partial_{G^*}(A_j)$ are exactly the vertices occupied by searchers after the last move of $\mathcal{S}_{m_j}$, where m_j is the index of the edge in $(A \cap E') \setminus (A_{j-1} \cap E')$, where A is the first crucial set of $\mathcal{E}$ such that $A_j \subseteq A$.

Proof of Claim 6. Clearly, this is true for $A_1 = E^{\text{in}}$. Assume that it holds for $A_{j'}$.

We will show that the vertices in $\partial_{G^*}(A_{j'+1})$ are exactly the vertices occupied by searchers after the last move of $\mathcal{S}_{m_{j'+1}}$.

If $A_{j'+1}$ is not crucial then $\partial_{G^*}(A_{j'+1}) \subseteq \partial_{G^*}(A_{j'})$ and $m_{j'+1} = m_{j'}$, therefore Claim 6 holds.

Now, if $A_{j'+1}$ is crucial and $\{e_{m_{j'+1}}\} = (A_{j'+1} \cap E') \setminus (A_{j'} \cap E')$, then $v_{m_{j'+1}} \in \partial_{G^*}(A_{j'})$ and, therefore, must be occupied by a searcher. We distinguish three cases:

Case 1. If $v_{m_{j'+1}} \in \partial_{G^*}(A_{j'+1})$ and $u_{m_{j'+1}} \in \partial_{G^*}(A_{j'+1})$, then $\partial_{G^*}(A_{j'+1}) = \partial_{G^*}(A_{j'}) \cup \{u_{m_{j'+1}}\}$ and the first move in $\mathcal{S}_{m_{j'+1}}$ will be $\mathsf{p}(u_{m_{j'+1}})$.

Case 2. If $v_{m_{j'+1}} \in \partial_{G^*}(A_{j'+1})$ and $u_{m_{j'+1}} \notin \partial_{G^*}(A_{j'+1})$ then $\partial_{G^*}(A_{j'+1}) = \partial_{G^*}(A_{j'})$.

Case 3. If $v_{m_{j'+1}} \notin \partial_{G^*}(A_{j'+1})$, then $\partial_{G^*}(A_{j'+1}) = (\partial_{G^*}(A_{j'}) \setminus \{v_{m_{j'+1}}\}) \cup \{u_{m_{j'+1}}\}$, and the first move in $\mathcal{S}_{m_{j'+1}}$ will be $\mathsf{s}(v_{m_{j'+1}}, u_{m_{j'+1}})$.

Observe that in all three cases the Claim 6 holds. □

Let $V_{\mathcal{S}}(i)$ be the set of vertices already visited by searchers after the i-th move of $\mathcal{S}$, and let $V_{\mathcal{S}} = \langle V_{\mathcal{S}}(1), \ldots, V_{\mathcal{S}}(r) \rangle$. Notice that this sequence is monotone and that if the i-th move belong to the subsequence $\mathcal{S}_j$, then $V_{\mathcal{S}}(i) = V(A_{l_{u_j}})$. We must next prove the following:

Claim 7. For every $i \in \{1, \ldots, |\mathcal{S}|\}$, all edges of $G^*[V_{\mathcal{S}}(i)]$ are clean.

Proof of Claim 7. Clearly, the claim is true for $i \in \{1, \ldots, 2 \cdot |S^{\text{in}}|\}$. Assume that it holds for some $i \in 2 \cdot |S^{\text{in}}| + 1, \ldots, r$, we will show that all edges of $G^*[V_{\mathcal{S}}(i+1)]$ are clean. We must distinguish three cases about the $(i+1)$-th move:

Case 1. It is a removal, say $\mathsf{r}(u)$. Notice that $G^*[V_{\mathcal{S}}(i+1)] = G^*[V_{\mathcal{S}}(i)]$, therefore the Claim will not be true if $\mathsf{r}(u)$ is a recontamination move. In this case, there exist an edge connecting u with a vertex not in $V_{\mathcal{S}}(i)$, say v. As $u \in \partial_{G^*}(A_{l_{u_j}-1}) \setminus \partial_{G^*}(A_{l_{u_j}})$, for some $j \in \{1, \ldots, |V^*|\}$, all edges with u as endpoint must belong to $A_{l_{u_j}}$, therefore $\{u, v\} \in A_{l_{u_j}}$. But $V_{\mathcal{S}}(i) = V(A_{l_{u_j}})$, a contradiction.

Case 2. It is a placement of searcher say $\mathsf{p}(u)$. By the definition of $\mathcal{S}$, there exist an edge $\{u, v\}$, where v is a vertex in $V_{\mathcal{S}}(i)$. Notice that, according to our search game, all such edges are clean after $\mathsf{p}(u)$, thus all edges of $G^*[V_{\mathcal{S}}(i+1)]$ are clean.

Case 3. It is a slide, say $\mathsf{s}(v_j, u_j)$, for some $j \in \{1, \ldots, |V^*|\}$. As in the previous case, $G^*[V_{\mathcal{S}}(i+1)]$ contains all edges of $G^*[V_{\mathcal{S}}(i)]$ and additional all edges with u_j as the first endpoint and a vertex $v \in V_{\mathcal{S}}(i)$ as the other. According to our search game, after the i-th move there must be searcher in v_j, therefore due to Claim 6, $v_j \in \partial_{G^*}(A_{l_{u_j}})$. Notice that, the Claim will not be true if $\mathsf{s}(v_j, u_j)$ is a recontamination move, i.e., there exist an edge connecting v_j with a vertex, say u, not in $V_{\mathcal{S}}(i) = V(A_{l_{u_j}})$. As $v_j \notin \partial_{G^*}(A_{l_{u_j}})$, all edges with u as endpoint must belong to $A_{l_{u_j}}$, therefore $\{v_j, u\} \in A_{l_{u_j}}$, a contradiction.

In all three cases we saw that after the $(i+1)$-th move of S all edges of $G^*[V_{\mathcal{S}}(i+1)]$ are clean, therefore Claim 7 is true. □

We will now prove that $\mathcal{S}$ is a (S_1, S_2)-complete strategy for G^*.

Clearly, Condition (i) holds for every strategy starting with $\mathcal{S}_0$. Moreover, Condition (ii) holds as v^{out} is not a vertex of V^* and therefore, no placement on u^{out} or sliding towards u^{out} appears in $\mathcal{S}$. Notice that, according to Claim 7, for every $i \in \{1, \ldots, |V^*|-1\}$, $E(\mathcal{S}, |\mathcal{S}_0 \circ \cdots \circ \mathcal{S}_{i-1}|+1) = \cdots = E(\mathcal{S}, |\mathcal{S}_0 \circ \cdots \circ \mathcal{S}_{i-1}| + |\mathcal{S}_i|) = A_{l_{u_{i+1}}-1}$, and that for $i = |V^*|$, $E(\mathcal{S}, |\mathcal{S}_0 \circ \cdots \circ \mathcal{S}_{|V^*|}|) = A_r$, therefore Condition (iii) holds.

By the definition of $\mathcal{S}$, it is clear that $\mathcal{S}$ is a connected search strategy, moreover, according to Claim 7, $\mathcal{S}$ is monotone.

It remains to prove that $\mathcal{S}$ has cost at most k. For the first $2|S^{\text{in}}|$ moves, we use $|S^{\text{in}}| = \mathsf{cost}_{G^*}(\mathcal{E}, 1) \leq k$ searchers. Assume that after j moves exactly k searchers are occupying vertices of G^* and that the $(j+1)$-th move is $\mathsf{p}(u_i)$, for some $i \in \{i, \ldots, |V^*|\}$. Then the vertices in $\partial_{G^*}(A_{l_{u_i}-1})$ are exactly the vertices occupied by the k searchers, therefore $|\partial_{G^*}(A_{l_{u_i}-1})| = k$. Observe that, if u_i is not pendant, then $\partial_{G^*}(A_{l_{u_i}}) = \partial_{G^*}(A_{l_{u_i}-1}) \cup \{u_i\}$, therefore $|\partial_{G^*}(A_{l_{u_i}})| = k+1$, a contradiction and if u_i is pendant then $\partial_{G^*}(A_{l_{u_i}}) = \partial_{G^*}(A_{l_{u_i}-1})$ and the cost of $\mathcal{E}$ at position l_{u_i} is $|\partial_{G^*}(A_{l_{u_i}})| + 1 = k+1$, again a contradiction. Thus, for every move of $\mathcal{S}$ at most k searchers are occupying vertices of G^*. □

9.1.3 Contractions

As we consider rooted graphs instead of regular graphs, we need to give new definitions for the contraction and minor relations, that will still be consistent with the definitions we gave in Chapter 6. We may define them for rooted graphs at first, but then, by taking $S_1^{\text{in}} = S_1^{\text{out}} = \emptyset$, we will restrict them to graphs.

Definition 9.1.13. Let $(G_1, S_1^{\text{in}}, S_1^{\text{out}})$ and $(G_2, S_2^{\text{in}}, S_2^{\text{out}})$ be rooted graphs. We say that $(G_1, S_1^{\text{in}}, S_1^{\text{out}})$ is *a (rooted) contraction* of $(G_2, S_2^{\text{in}}, S_2^{\text{out}})$ and we denote this fact by $(G_1, S_1^{\text{in}}, S_1^{\text{out}}) \leq_{\text{c}} (G_2, S_2^{\text{in}}, S_2^{\text{out}})$ if there exists a surjection $\phi : V(G_2) \to V(G_1)$ such that:

(1) for every vertex $v \in V(G_1)$, $G_2[\phi^{-1}(v)]$ is connected,

(2) for every two distinct vertices $u, v \in V(G_1)$, it holds that $\{v, u\} \in E(G_1)$ if and only if the graph $G_2[\phi^{-1}(v) \cup \phi^{-1}(u)]$ is connected,

(3) $\phi(S_2^{\text{in}}) = S_1^{\text{in}}$, and

(4) $\phi(S_2^{\text{out}}) = S_1^{\text{out}}$.

We will often write $(G_1, S_1^{\text{in}}, S_1^{\text{out}}) \leq_{\text{c}}^{\phi} (G_2, S_2^{\text{in}}, S_2^{\text{out}})$ to make clear that the function certifying the contraction relation is ϕ. Notice that G_1 is a *contraction* of G_2 if $(G_1, \emptyset, \emptyset) \leq_{\text{c}} (G_2, \emptyset, \emptyset)$.

We can define the *(rooted) minor* relation for two rooted graphs by removing in the second property the demand that if $\{u, v\} \notin E(G_1)$ then $G_2[\phi^{-1}(v) \cup \phi^{-1}(u)]$ is not connected. We denote the minor relation by $(G_1, S_1^{\text{in}}, S_1^{\text{out}}) \leq_{\text{m}} (G_2, S_2^{\text{in}}, S_2^{\text{out}})$. Notice that G_1 is a *minor* of G_2 if $(G_1, \emptyset, \emptyset) \leq_{\text{m}} (G_2, \emptyset, \emptyset)$.

Lemma 9.1.4. If $(G_1, S_1^{\text{in}}, S_1^{\text{out}})$ and $(G_2, S_2^{\text{in}}, S_2^{\text{out}})$ are rooted graphs and $(G_1, S_1^{\text{in}}, S_1^{\text{out}}) \leq_{\text{c}} (G_2, S_2^{\text{in}}, S_2^{\text{out}})$, then $\mathbf{cmp}(G_1, S_1^{\text{in}}, S_1^{\text{out}})) \leq \mathbf{cmp}(G_2, S_2^{\text{in}}, S_2^{\text{out}}))$.

Proof. Suppose that $\mathcal{E} = \langle A_1, \ldots, A_r \rangle$ is a monotone $(E_2^{\text{in}}, E_2^{\text{out}})$-expansion of $G_2^* = \mathbf{enh}(G_2, S_2^{\text{in}}, S_2^{\text{out}})$ with cost at most k. Our target is to construct a monotone $(E_1^{\text{in}}, E_1^{\text{out}})$-expansion of $G_1^* = \mathbf{enh}(G_1, S_1^{\text{in}}, S_1^{\text{out}})$ with cost at most k.

Let ϕ be a function where $(G_1, S_1^{\text{in}}, S_1^{\text{out}}) \leq_{\text{c}}^{\phi} (G_2, S_2^{\text{in}}, S_2^{\text{out}})$. We consider an extension ψ of ϕ that additionally maps u_2^{in} to u_1^{in} and u_2^{out} to u_1^{out}. Notice that the construction of ψ yields the following:

$$(G_1^*, S_1^{\text{in}} \cup \{u_1^{\text{in}}\}, S_1^{\text{out}} \cup \{u_1^{\text{out}}\}) \leq_{\text{c}}^{\phi} (G_2^*, S_2^{\text{in}} \cup \{u_2^{\text{in}}\}, S_2^{\text{out}} \cup \{u_2^{\text{out}}\})$$

Given an edge $f = \{x, y\} \in E(G_1)$ we consider the set E_f containing all edges of G_2 with one endpoint in $\psi^{-1}(x)$ and one endpoint in $\psi^{-1}(y)$. We now pick – arbitrarily – an edge in E_f and we denote it by e_f. We also set $E' = \{e_f \mid f \in E(G_1)\}$. Then it is easy to observe that $\mathcal{E}' = \langle A_1 \cap E', \ldots, A_r \cap E' \rangle$ is a connected expansion of G_1^* and that the cost of $\mathcal{E}'$ at step i is no bigger than the cost of $\mathcal{E}$ at the same step, where $i \in \{1, \ldots, r-1\}$. □

Lemma 9.1.5. If G_1 and G_2 are two graphs and $G_1 \leq_{\text{c}} G_2$, then $\mathbf{cms}(G_1) \leq \mathbf{cms}(G_2)$.

Proof. First observe that, if this is the case, any contraction of G_2 can be derived by applying a finite number of edge-contractions of some edges in $E(G_2)$.

It suffices to prove that the lemma holds if G_1 is obtained by the contraction of edge $e = \{u, v\} \in E(G_1)$ to vertex x_{uv}. Let $\mathcal{S}$ be a connected search strategy for G_2 that – in any step – uses at most k searchers. Based on $\mathcal{S}$ we will construct a search strategy $\mathcal{S}'$ for G_1. Let i be an integer in $\{1, \ldots, |\mathcal{S}|\}$. We distinguish eight cases:

Case 1: If the i-th move of $\mathcal{S}$ is $\mathsf{p}(x)$ for some vertex $x \notin \{u, v\}$, then the next move of $\mathcal{S}'$ will be $\mathsf{p}(x)$.

Case 2: If the i-th move of $\mathcal{S}$ is $\mathsf{r}(x)$ for some vertex $x \notin \{u, v\}$, then the next move of $\mathcal{S}'$ will be $\mathsf{r}(x)$.

Case 3: If the i-th move of $\mathcal{S}$ is $\mathsf{s}(x, y)$ for some vertices $x, y \notin \{u, v\}$, then the next move of $\mathcal{S}'$ will be $\mathsf{s}(x, y)$.

Case 4: If the i-th move of $\mathcal{S}$ is $\mathsf{p}(u)$ or $\mathsf{p}(v)$, then the next move of $\mathcal{S}'$ will be $\mathsf{p}(x_{uv})$.

Case 5: If the i-th move of $\mathcal{S}$ is $\mathsf{r}(u)$ or $\mathsf{r}(v)$, then the next move of $\mathcal{S}'$ will be $\mathsf{r}(x_{uv})$.

Case 6: If the i-th move of $\mathcal{S}$ is $\mathsf{s}(z, u)$ or $\mathsf{p}(z, v)$ for some vertex z, then the next move of $\mathcal{S}'$ will be $\mathsf{s}(z, x_{uv})$.

Case 7: If the i-th move of $\mathcal{S}$ is $\mathsf{s}(u, z)$ or $\mathsf{p}(v, z)$ for some vertex z, then the next move of $\mathcal{S}'$ will be $\mathsf{s}(x_{uv}, z)$.

Case 8: If the i-th move of $\mathcal{S}$ is $\mathsf{s}(u, v)$ or $\mathsf{p}(v, u)$, then the next move of $\mathcal{S}'$ will be defined according the lateral cases from the $(i+1)$-th move of $\mathcal{S}$.

Observe that $\mathcal{S}'$ is a complete search strategy for G_1. Furthermore, as $\mathcal{S}$ is connected, $\mathcal{S}'$ must also be connected. Finally, it is clear that $\mathcal{S}'$ – at any step – uses at most k searchers, thus $\mathbf{cms}(G_1) \leq k$. □

9.1.4 Cut-vertices and blocks

Let us fix some more notation. Recall that the *blocks* of a graph G are its 2-connected components. If the removal of an edge in G increases the number of its connected components, then it is called a *bridge*. In what follows we will consider the subgraphs of G induced by the endpoints of

bridges as blocks and we call them *trivial blocks* of G.

Definition 9.1.14. Let G be a graph and B a block of G. A *cut-vertex* of G is a vertex such that $G \setminus x$ has more connected components than G. A *cut-vertex of* B is a cut-vertex of G belonging to $V(B)$.

The following two definitions may be highly technical but are very helpful for the proofs of Section 9.2.

Definition 9.1.15. Let G bet a graph and let $x \in V(G)$. We define

$$\mathcal{C}_G(x) = \{(x, G[V(C) \cup \{x\}]) \mid C \text{ is a connected component of } G \setminus x\}.$$

Definition 9.1.16. Let B be a block of G and let x be a cut-vertex of B. We denote by $C_G(x, B)$ the (unique) graph in $\mathcal{C}_G(x)$ that contains B as a subgraph and by $\overline{\mathcal{C}}_G(x, B)$ the graphs in $\mathcal{C}_G(x)$ that do not contain B.

Recall the definition of graph embeddings in $\mathbb{R}^d$ (Definition 5.1.4). A *plane embedding* $\mathcal{E}_2(G) = (f, \mathcal{C})$ of a graph G is an embeddings in $\mathbb{R}^2$ such that the interior of each curve in $\mathcal{C}$ contains neither a point $f(u)$, for some $u \in V(G)$, nor a point of another curve in $\mathcal{C}$. Notice that, given a plane embedding of G the set $\mathbb{R}^2 \setminus \mathcal{E}_2(G)$ defined when we subtract from $\mathbb{R}^2$ the points $\{f(u) \mid u \in V(G)\}$ corresponding to the vertices of G, and the points $\bigcup_{c \in \mathcal{C}} C$ of the curves corresponding to the edges of G, is open. The regions of $\mathbb{R}^2 \setminus \mathcal{E}_2(G)$ are the faces of $\mathcal{E}_2(G)$. We define the outer face and the inner faces of $\mathcal{E}_2(G)$ in the same way we did in Definition 2.5.2.

Definition 9.1.17. A graph G is *outerplanar* if there is a plane embedding of G such that all its vertices are incident to the outer face.

This embedding, when it exists, is unique up to homeomorphism.

From now on, each outerplanar graph G is accompanied with such an embedding, say $\mathcal{E}_2(G) = (f, \mathcal{C})$. We will refer to the elements of $f(V)$

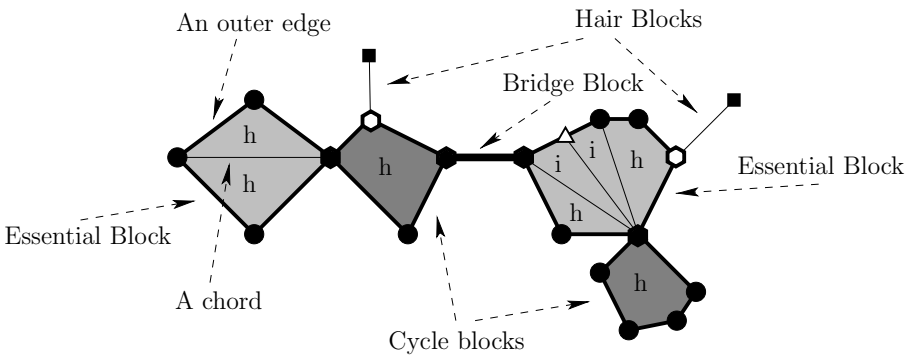


Figure 9.3: An outerplanar graph and its blocks. The cut-vertices are hexagonal and the outer vertices are squares. Inner and haploid faces are denoted by "i" and "h" respectively. There are, in total, four inner vertices (all belonging to the essential block on the right) and, among them only the triangular one is not an haploid vertex. The white hexagonal vertices are the light cut-vertices while the rest of the hexagonal vertices are the heavy ones.

and $\mathcal{C}$ as the vertices and edges of $\mathcal{E}_2(G)$ respectively and, to keep thinks simple, we will identify them with the "real" vertices and edges of G.

An edge $e \in E(G)$ is called *outer edge of* G, if it is incident to the outer face of G, otherwise it is called a *chord of* G.

A face F of an outerplanar graph that is different than the outer face, is called *haploid* if and only if at most one edge incident to F is a chord, otherwise F is a *inner* face. A vertex $u \in V(G)$ is *haploid* if it is incident to an haploid face and *inner* if it is incident to an inner face (notice that some vertices can be both inner and haploid). A vertex of G that is not inner or haploid is called *outer*. We call a chord *haploid* if it is incident to an haploid face. Non-haploid chords are called *internal chords*.

OBSERVATION 9.1.1. A block of a connected outerplanar graph with more than one edge can be one of the following.

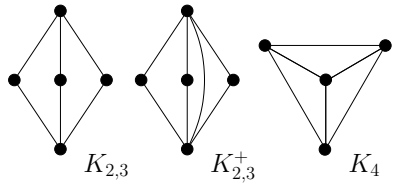


Figure 9.4: The set $\mathcal{O}_1$.

- A *hair block*: It is a trivial block containing exactly one vertex of degree 1 in G.
- A *bridge block*: It is a trivial block that is not a hair-block.
- A *cycle block*: If it is a chordless non-trivial block.
- An *essential block*: If it is a non-trivial block with at least one chord.

Let G be a connected outerplanar graph with more than one edge. Given a cut-vertex c of G, we say that c is *light* if it is the (unique) cut-vertex of exactly one hair block. If a cut-vertex of G is not light then it is *heavy* (see Figure 9.10 for an example).

It is known that the class of outerplanar graphs is closed under the relations $\leq_{\mathrm{m}}, \leq_{\mathrm{c}}$ and, without much effort, one can derive from Theorem 2.5.2 the following.

Theorem 9.1.1. *A graph is outerplanar if and only if* $K_4 \not\leq_{\mathrm{m}} G$ *and* $K_{2,3} \not\leq_{\mathrm{m}} G$.

Let $K^+_{2,3}$ be the graph obtained by $K_{2,3}$ after connecting the two vertices of degree 3 (Figure 9.4).

Lemma 9.1.6. If $\mathcal{H}$ is the class of all outerplanar graphs, then $\mathsf{obs}_{\leq_{\mathrm{c}}}(\mathcal{H}) = \mathcal{O}_1$, where $\mathcal{O}_1$ is shown in Figure 9.4.

Proof. Observe that the graphs in $\mathcal{O}_1$ cannot be embedded in the plane in such a way that all of its vertices are incident to a single face, and, therefore, neither the graphs in $\mathcal{O}_1$, neither the graphs that contain as a contraction a graph in $\mathcal{O}_1$, can be outerplannar.

To complete the proof, one must show that every non-outerplannar graph can be contracted to a graph in $\mathcal{O}_1$. Let G be non-outerplannar, then $K_4 \leq_{\mathrm{m}} G$ or $K_{2,3} \leq_{\mathrm{m}} G$. Clearly, as K_4 is a clique, $K_4 \leq_{\mathrm{m}} G$ implies that $K_4 \leq_{\mathrm{c}} G$. Suppose now that $K_{2,3} \leq_{\mathrm{m}} G$. Let V_x, V_y, V_1, V_2, V_3 be the vertex sets of the connected subgraphs of G that are contracted towards creating the vertices of $K_{2,3}$ (V_x and V_y are contracted to vertices of degree 3). If there is no edge in G between two vertices in V_a and V_b for some $(a, b) \in \{(x, y), (1, 2), (2, 3), (1, 3)\}$ then $K_{2,3} \leq_{\mathrm{c}} G$. If the only such edge is between V_x and V_y then $K^+_{2,3} \leq_{\mathrm{c}} G$ and in any other case, $K_4 \leq_{\mathrm{c}} G$. □

9.2 Obstructions for Graphs With cms/cmms at Most 2

In this Section we define a "candidate" contraction obstruction set for graphs with connected monotone mixed search number at most 2, namely the set $\mathcal{D}^1$ defined below. This set consists of 177 graphs. Then, we prove a series of lemmata, each providing some information about the structure of the graphs that can be searched with at most two searchers, using a connected and monotone mixed search strategy. Combining all these lemmata we will prove that $\mathcal{D}^1$ is the correct obstruction set.

Definition 9.2.1. Let $\mathcal{D}^1 = \mathcal{O}_1 \cup \cdots \cup \mathcal{O}_{12}$ where $\mathcal{O}_1$ is depicted in Figure 9.4, $\mathcal{O}_2, \ldots, \mathcal{O}_9$ are depicted in Figure 9.5 and $\mathcal{O}_{10}$ and $\mathcal{O}_{11}$ and $\mathcal{O}_{12}$ are constructed as follows.

$\mathcal{O}_{10}$: Contains every graph that can be constructed by taking three dis-

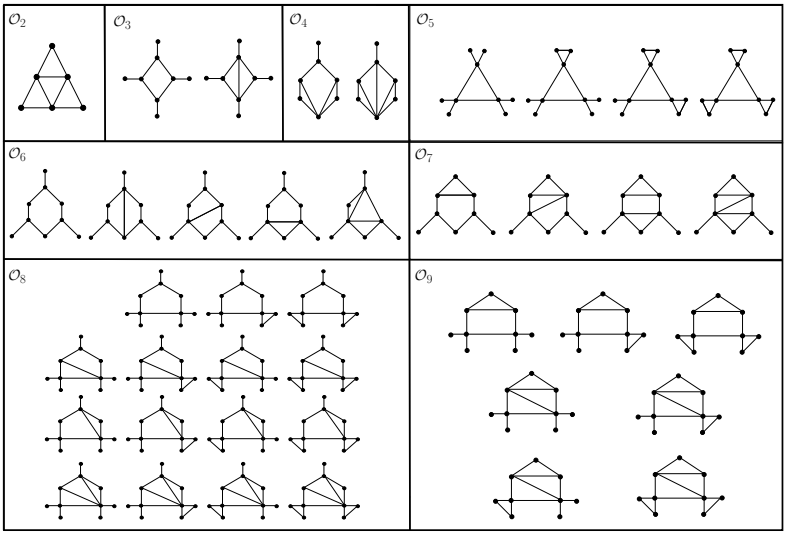


Figure 9.5: Some of the sets of graphs in Definition 9.2.1.

joint copies of some graphs in Figure 9.10 and then, identify the vertices denoted by v in each of them to a single vertex. There are – in total – 35 graphs generated in this way.

$\mathcal{O}_{11}$: Contains every graph that can be constructed by taking two disjoint copies of some graphs in Figure 9.14 and then, identify the vertices denoted by v in each of them to a single vertex. There are – in total – 78 graphs generated in this way.

$\mathcal{O}_{12}$: Contains every graph that can be constructed by taking two disjoint copies of some graphs in Figure 9.15 and then, identify the vertices denoted by v in each of them to a single vertex. There are – in total – 21 graphs generated in this way.

Notice that, $\mathcal{D}^1$ indeed contains 177 graphs.

9.2.1 Proof strategy

We will prove that $\mathcal{D}^1$ is the obstruction set we are looking for, in two steps. One easy (Lemma 9.2.1) and one much more involved (Lemma 9.2.2). A very important part for the proof of the second step is to distinguish "important blocks" from non "important ones", namely *central* and *extremal blocks* from *fans*, and then define *directional obstructions*.

Lemma 9.2.1. $\mathcal{D}^1 \subseteq \mathsf{obs}_{\leq_c}(\mathbf{cmp}, 2)$.

Proof. From Lemma 9.1.4, it is enough to check that for every $G \in \mathcal{D}^1$, the following two conditions are satisfied

(i) $\mathbf{cmp}(G) \geq 3$, and

(ii) for every edge e of G it holds that $\mathbf{cmp}(G/e) \leq 2$.

One can verify that this is correct by inspection, as this concerns only a finite amount of graphs and, for each of them, there exists a finite number of edges to contract. □

Lemma 9.2.2. $\mathcal{D}^1 \supseteq \mathsf{obs}_{\leq_c}(\mathbf{cmp}, 2)$.

The rest of this Section is devoted to the proof of Lemma 9.2.2. For this, our strategy is to consider the set

$$\mathcal{Q} = \mathsf{obs}_{\leq_c}(\mathbf{cmp}, 2) \setminus \mathcal{D}^1$$

and prove that $\mathcal{Q} = \emptyset$ (Lemma 9.2.15). To achieve this, as mentioned before, we have to prove a series of structural results. Their proofs use the following three fundamental properties of the set $\mathcal{Q}$.

Lemma 9.2.3. Let $G \in \mathcal{Q}$. Then the following hold.

(i) $\mathbf{cmp}(G) \geq 3$.

(ii) If H is a proper contraction of G, then $\mathbf{cmp}(G) \leq 2$.

(iii) G does not contain any of the graphs in $\mathcal{D}^1$ as a contraction.

Proof. Properties (i) and (ii) hold because $G \in \mathsf{obs}_{\leq_c}(\mathbf{cmp}, 2)$. For property (iii) suppose, to the contrary, that G contains some graph in $H \in \mathcal{D}^1$ as a contraction. From Lemma 9.2.1, $H \in \mathsf{obs}_{\leq_c}(\mathbf{cmp}, 2)$. Clearly, H is different than G as $\mathcal{Q}$ does not contain members of $\mathcal{D}^1$. Therefore, H is a proper contraction of G and, from property (ii), $\mathbf{cmp}(H) \leq 2$. This contradicts to the fact that $H \in \mathsf{obs}_{\leq_c}(\mathbf{cmp}, 2)$ and thus $\mathbf{cmp}(H) \geq 3$. □

9.2.2 Basic structural properties

Lemma 9.2.4. Let $G \in \mathcal{Q}$. The following hold:

1. G is outerplanar.
2. Every light cut-vertex of G has degree at least 3.
3. Every essential block B of G, has exactly two haploid faces.
4. Every block of G, has at most 3 cut-vertices
5. Every cut-vertex of a non-trivial block of G is an haploid vertex.
6. Every block of G contains at most 2 heavy cut-vertices.
7. If a block of G has 3 cut-vertices, then there are two, say x and y, of these vertices that are not both heavy and are connected by an haploid edge.
8. If an essential block of G with haploid faces F_1 and F_2 has two heavy cut-vertices, then we can choose one, say c_1, of these two heavy cut-vertices so that it is incident to F_1 and one say c_2 that is incident to F_2. Moreover, this assignment can be done in such a way

that if there is a third light cut-vertex c_3, adjacent to one, say c_1, of c_1, c_2, then c_3 is incident to F_1 as well.

Proof. 1. By the third property of Lemma 9.2.3, G cannot be contracted to a graph in $\mathcal{O}_1$ and therefore, from Lemma 9.1.6, G must be outerplannar.

2. Let c be a light cut-vertex of a block B in G, with degree 2 (notice that, as c is a cut-vertex, c cannot have degree 1 or 0). That means that c belongs to a path with at least two edges, the hair block B and an edge say e. Observe that $\mathbf{cmp}(G/B) = \mathbf{cmp}(G)$, contradicting to the second property of Lemma 9.2.3.

3. Let B be an essential block of G. As it is essential, it has at least one chord, therefore it has at least 2 haploid faces. Assume, that B has at least 3 haploid faces. Choose 3 of them, say F_1, F_2 and F_3 (see Figure 9.6). Let $S \subseteq E(B)$ be the set of all chords incident to B. Contract in G all edges in $E(G) \setminus S$ not belonging to those faces. Then, for each of the three faces, contract all but two edges not in S that are incident to F_1, F_2 and F_3 and notice that the obtained graph is the graph in $\mathcal{O}_2$, a contradiction to the third property of Lemma 9.2.3.

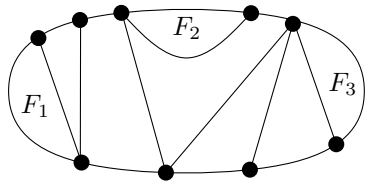


Figure 9.6: An example for the proof of Lemma 9.2.4.3.

4. Let B be a block of G containing more than 3 cut-vertices. Choose four of them, say c_1, c_2, c_3 and c_4. Let $S \subseteq E(B)$ be the set of all chords incident to B (see Figure 9.7). Contract all edges in $E(G) \setminus S$ not having

an endpoint in $\{c_1, c_2, c_3, c_4\}$. Then, contract all edges $e \in E(B) \setminus S$ such that $e \nsubseteq \{c_1, c_2, c_3, c_4\}$ and all edges not in $E(B)$, except from one for each of the cut-vertices. Notice that the obtained graph belongs to $\mathcal{O}_3$, a contradiction to the third property of Lemma 9.2.3.

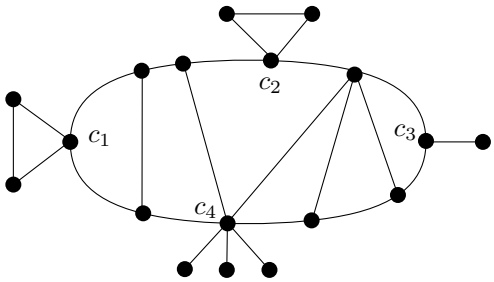


Figure 9.7: An example for the proof of Lemma 9.2.4.4.

5. Let B be a block of G containing a cut-vertex c that is not haploid and let $S \subseteq E(B)$ be the set of all chords incident to B (see Figure 9.8). Contract all edges in $E(G) \setminus E(B)$ not having c as endpoint and all edges in $E(B) \setminus S$ not having c as endpoint, except from two edges for each of the haploid faces. Then contract all edges not in $E(B)$ with c as endpoint, except for one. Notice that the obtained graph belongs to $\mathcal{O}_4$, a contradiction to the third property of Lemma 9.2.3.

6. Let B be a block of G containing three heavy cut-vertices, say c_1, c_2 and c_3 (see Figure 9.9). We contract all edges in B except from 3 so that B is reduced to a triangle T with vertices c_1, c_2 and c_3. Then, in the resulting graph H, for each $c_i, i \in \{1, 2, 3\}$, in $\mathcal{C}_H(c_i) \setminus \{T\}$ contains either a non trivial block or at least two hair blocks. In any case, H can be further contracted to one of the graphs in $\mathcal{O}_5$ a contradiction to the third property of Lemma 9.2.3.

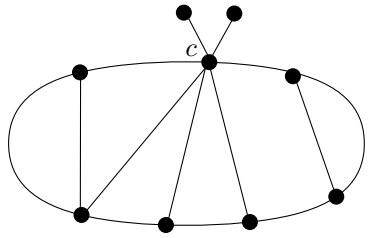


Figure 9.8: An example for the proof of Lemma 9.2.4.5.

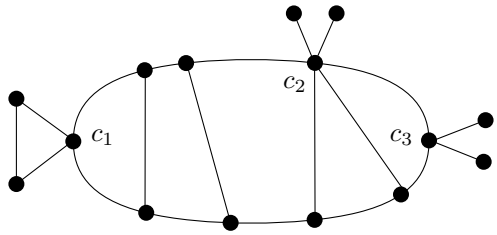


Figure 9.9: An example for the proof of Lemma 9.2.4.6.

7. Let $\{x, y, z\}$ be three cut-vertices of a (not-trivial) block B. If no two of them are connected by an outer edge, then contract all blocks of G, except B, to single edges, then contract all outer edges of B that do not have an endpoint in $\{x, y, z\}$ and continue contracting hair blocks with a vertex of degree ≥ 4, as long as this is possible. This creates either a graph in $\mathcal{O}_6$ or a graph that after the contraction of a hair block makes a graph in $\mathcal{O}_7$ or a graph that after the contraction of two hair blocks is a graph makes a graph in $\mathcal{O}_4$ and, in any case, we have a contradiction to the third property of Lemma 9.2.3. We contract G to a graph H as follows:

- If for some $w \in \{x, y, z\}$ the set $\overline{\mathcal{C}}_G(w, B)$ contains at least two elements, then contract the two of them to a pendant edge (that will have w as an endpoint) and the rest of them to w.

- If for some $w \in \{x, y, z\}$ the set $\overline{\mathcal{C}}_G(w, B)$ contains only one ele-

ment that is not a hair, then contact it to a triangle (notice that this is always possible because of 2).

Case 1. $|V(B)| \in \{3, 4\}$. Then because of 6, one, say x of $\{x, y, z\}$ is non-heavy and there is an outer edge connecting x with one, say y, vertex in $\{x, y, z\}$. Then x, y is the required pair of vertices.

Case 2. $|V(B)| > 4$ and there is at most one outer edge e with endpoints from $\{x, y, z\}$ in H. W.l.o.g., we assume that $e = \{x, y\}$. Notice e is a haploid edge, otherwise H can be contracted to the 5th graph in $\mathcal{O}_6$. Moreover at least one of x, y is non-heavy, otherwise H can be contracted to one of the graphs in $\mathcal{O}_8 \cup \mathcal{O}_9$ (recall that B may have one or two haploid faces).

Case 3. There are two outer edges with endpoints from $\{x, y, z\}$. W.l.o.g., we assume that these edges are $\{x, y\}$ and $\{y, z\}$. One, say $\{x, y\}$, of $\{x, y\}$, $\{y, z\}$ is haploid, otherwise H can be contracted to some graph in $\mathcal{O}_4$. If $\{x, y\}$ has a light endpoint, then we are done, otherwise, from 6, z is light. In this remaining case, if if $\{z, y\}$ is haploid, then it is also the required edge, otherwise H can be contracted to a graph in $\mathcal{O}_9$.

8. Let x and y be two heavy cut-vertices vertices of B. From 5, x, y are among the vertices that are incident to the faces F_1 and F_2. Suppose, in contrary, that for some face, say $F \in \{F_1, F_2\}$, there is no cut vertex in $\{x, y\}$ that is incident to F. Then G can be contracted to one of the graphs in $\mathcal{O}_9$. This is enough to prove the first statement except from the case where x and y are both lying in both haploid faces and there is a third light cut-vertex z incident to some, say x, of x, y. In this case, x is assigned the face where z belongs and y is assigned to the other. □

Let $G \in \mathcal{Q}$ and let B be a block of G. Let also S be the set of cut vertices of G that belong to B. According to Lemma 9.2.4, we can define a rooted graph $\mathbf{G}_B = (B, X, Y)$ such that:

- $\{X, Y\}$ is a partition of S, where X and Y are possibly empty.
- If B has a chord, then all vertices in X and Y are haploid.
- $|X| \leq 1$ and $|Y| \leq 2$.
- If $|Y| = 2$, then its vertices are connected with an edge e and one of them is light and. Moreover, in the case where B has a chord then e is haploid.
- If B has a chord, we name the haploid faces of B by F_1 and F_2 such that all vertices in X are incident to F_1 and all vertices od Y are incident to F_2.

LEMMA 9.2.5. Let $G \in \mathcal{Q}$ and let B be a block of G. Then $\mathbf{cmp}(\mathbf{G}_B) \leq 2$.

Proof. We examine the – non-trivial – case where B is a non-trivial block and contains two haploid faces F_1 and F_2. As B is 2-connected and outerplanar, all vertices of $V(B)$ belong to the unique hamiltonian cycle of B, say C. Our proof is based on the fact that there are exactly two haploid faces and this gives a sense of direction on how the search should be performed. To make this formal, we create an ordering $\mathcal{A}$ of the edges of $E(B)$ using the following procedure.

1. **if** $X \neq \emptyset$, **then**
2. $\quad Q \leftarrow X$,
3. $\quad$ **else**
4. $\quad Q \leftarrow \{x\}$ where x is an arbitrarily chosen vertex in the boundary of F_1.
5. $R \leftarrow Q$
6. $i \leftarrow 1$
7. **while** there is a vertex v in $V(B) \setminus R$ that is connected with some vertex, say u, in $Q \setminus Y$ whose unique neighbor in $V(B) \setminus R$ is v,
8. $\quad R \leftarrow R \cup \{v\}$

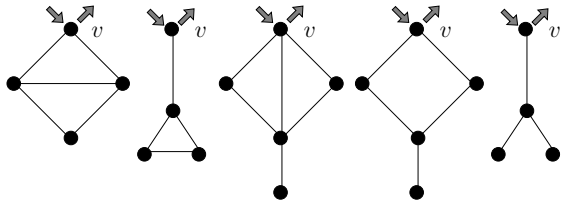


Figure 9.10: The set $\mathcal{A}$ contains five r-graphs, each of the form $(G, \{v\}, \{v\})$.

9. $Q \leftarrow (Q \setminus \{u\}) \cup \{v\}$
10. $e_i = \{u, v\}$
11. $i \leftarrow i + 1$
12. **if** $Q \in E(B)$, **then**
13. $e_i \leftarrow Q$,
14. $i \leftarrow i + 1$
15. **if** $Y \in E(B)$, **then**
16. $e_i \leftarrow Y$

Let $E^{\text{in}}, E^{\text{out}}$ be the edge extensions of $\mathbf{enh}(\mathbf{G}_B)$, and let $\mathsf{prefsec}(\mathcal{A}) = \langle A_0, \ldots, A_r \rangle$. It is easy to verify that $\mathcal{E} = \langle E^{\text{in}}, A_0 \cup E^{\text{in}}, \ldots, A_r \cup E^{\text{in}} \rangle$ is a monotone and connected $(E^{\text{in}}, E^{\text{out}})$-expansion of $\mathbf{enh}(\mathbf{G}_B)$, with cost at most 2. □

9.2.3 Fans

Let G be a graph and v be a vertex in $V(G)$. We denote by $\mathbf{G}^{(v)}$ the rooted graph $(G, \{v\}, \{v\})$ and we refer to it as the *graph G doubly rooted on v*.

Definition 9.2.2. A graph G, doubly rooted on some vertex v is a *fan* if none of the graphs in the set $\mathcal{A}$ depicted in Figure 9.10 is a contraction of the rooted graph $\mathbf{G}^{(v)}$, and G is outerplanar.

Fans will play an important role in this proof.

LEMMA 9.2.6. Let $\mathbf{G}^{(v)} = (G, \{v\}, \{v\})$ be a graph doubly rooted at some vertex v. If $\mathbf{G}^{(v)}$ is a fan, then $\mathbf{cmp}(\mathbf{G}^{(v)}) \leq 2$.

Proof. We claim first that if $\mathbf{G}^{(v)}$ is a fan, then the graph $G \backslash v$ is a collection of paths where each of them has at least one endpoint that is a neighbour of v. Indeed, if this is not correct, then some of the connected components of $G \setminus v$ would be contractible to either a K_3 or a $K_{1,3}$. In the first case, G is either non-outperlanar or $\mathbf{G}^{(v)}$ can be contracted to to the first two rooted graphs of Figure 9.10. In the second case G is either non-outeplanar or $\mathbf{G}^{(v)}$ the last three graphs of Figure 9.10. Moreover, if both endpoints of a path in the set of connected components $G \setminus v$ are non adjacent to v in G, then $\mathbf{G}^{(v)}$ can be contracted to the last rooted graph in Figure 9.10.

Let now $P_1, \ldots, P_r$ be the connected components of $G \setminus v$ and, for each $i \in \{1, \ldots, r\}$, let $\{v_1^i, \ldots, v_{j_i}^i\}$ be the vertices of P_i, ordered as in P_i, such that v_1^i is adjacent to v in G. Let u^{in} and u^{out} be the two vertices added in $\mathbf{enh}(\mathbf{G}^{(v)})$. If $e^{\text{in}} = \{v, u^{\text{in}}\}$ and $e^{\text{out}} = \{v, u^{\text{out}}\}$ then the edge expansions of $\mathbf{enh}(\mathbf{G}^{(v)})$ is $E^{\text{in}} = \{e^{\text{in}}\}$ and $E^{\text{out}} = \{e^{\text{out}}\}$.

For each $i \in \{1, \ldots, r\}$ we define the edge ordering

$$\mathcal{A}_i = \langle \{v, v_1^i\}, \{v_1^i, v_2^i\}, \{v, v_2^i\}, \{v_2^i, v_3^i\} \ldots, \{v_{j_i}^i, v\} \rangle,$$

then we delete from $\mathcal{A}_i$ the edges not in $E(G)$. Let $\mathcal{A}_i'$ be the orderings obtained after the edge deletions. We define $\mathcal{A} = \langle e^{\text{in}} \rangle \circ \mathcal{A}_1 \circ \cdots \circ \mathcal{A}_r$. Notice that $\mathsf{prefsec}(\mathcal{A})$ is a monotone and connected $(E^{\text{in}}, E^{\text{out}})$-expansion of $\mathbf{enh}(\mathbf{G}^{(v)})$ with cost at most 2. Therefore, it holds that $\mathbf{cmp}(\mathbf{G}^{(v)}) \leq 2$, as required. □

9.2.4 Spine-degree and central blocks

Given a graph G and a vertex v we denote by $\mathcal{C}_G^{(v)}$ the set of all graphs in $\mathcal{C}_G(v)$, each doubly rooted on v. The *spine-degree* of v is the number of

doubly rooted graphs in $\mathcal{C}_G^{(v)}$ that are not fans.

A cut-vertex of a graph G is called a *central cut-vertex*, if it has spine-degree greater than 1 and a block of G is called a *central block* if it contains at least 2 central cut-vertices.

Lemma 9.2.7. Let $G \in \mathcal{Q}$. The following hold:

1. All vertices of G have spine-degree at most 2.
2. None of the blocks of G contains more than 2 central cut-vertices.
3. G contains at least one central cut-vertex
4. There is a total ordering $B_1, B_2, \ldots, B_r$ ($r \geq 0$) of the central blocks of G and a total ordering $c_1, \ldots, c_{r+1}$ of the central cut-vertices of G such that, for $i \in \{1, \ldots, r\}$, the cut-vertices of B_i are c_i and c_{i+1}.

Proof. 1. Let v be a vertex of G with spine-degree at least 3. That means that there exist at least three subgraphs of G, doubly rooted at v, that can be contracted to some graph in $\mathcal{A}$. Therefore, G can be contracted to a graph in $\mathcal{O}_{10}$, a contradiction.

2. Suppose that B is a block of G containing 3 (or more) central cut-vertices, say c_1, c_2 and c_3. Construct the graph H by contracting all edges of B to a triangle T with $\{c_1, c_2, c_3\}$ as vertex set. As c_i is a central vertex, there is a rooted graph R_i in $\mathcal{C}_G(c_i)$ that contains some of the graphs in $\mathcal{A}$ as a rooted contraction. Next we apply the same contractions to H f or every $c_i, i \in \{1, 2, 3\}$ and then contract – to vertices – all blocks of H different than T and not contained in some R_i. It is easy to see that the resulting graph is a graph in $\mathcal{O}_5$, a contradiction.

3. Assume that G has no central cut-vertices. We distinguish two cases.

Case 1. There is a cut-vertex of G, say v, such that all rooted graphs in $\mathcal{C}_G^{(v)}$ are fans and let $\mathbf{G}_1, \ldots \mathbf{G}_r$ be these rooted graphs. From Lemmata 9.1.2 and 9.2.6, we conclude that $\mathbf{cmp}(G, \{v\}, \{v\}) = \mathbf{cmp}(\mathbf{glue}(\mathbf{G}_1, \ldots \mathbf{G}_r)) \leq 2$ and from Lemma 9.1.1, $\mathbf{cmp}(G) \leq 2$ contradicting to the first condition of Lemma 9.2.3.

Case 2. For every cut-vertex v of G, at least one of the rooted graphs in $\mathcal{C}_G^{(v)}$ is not a fan. We denote by H_v the corresponding non-fan rooted graph in $\mathcal{C}_G^{(v)}$ (this is unique due to the fact that v is non-central). Among all cut vertices, let x be one for which the set $V(G) \setminus V(H_x)$ is maximal. Let B be the block of H_x that contains x and let S be the set of the cut-vertices of G that belong to B (including x).

For every $y \in S$ we denote by $\mathcal{W}_y = \{\mathbf{W}_y^1, \ldots, \mathbf{W}_y^{r_y}\}$ the set of all rooted graphs in $\mathcal{C}_G^{(y)}$, except from the one, call it $\mathbf{R}_y$, that contains B. We also define $\mathbf{W}_y = \mathbf{glue}(\mathbf{W}_y^1, \ldots, \mathbf{W}_y^{r_y})$. We claim that all $\mathbf{W}_y, y \in S$ are fans. Indeed, if for some y, $\mathbf{W}_y$ is a non-fan, because y is not central, $\mathbf{R}_y$ should be a fan, contradicting the choice of x.

According to the above, the edges of G can be partitioned to those of the rooted graph $\mathbf{G}_B$ and the edges in the rooted graphs $\mathbf{W}_y,\ y \in S$. Let also $\mathbf{G}_B = (B, X, Y)$ and we assume that, if $Y = \{y_1, y_2\}$, then y_1 is light.

Notice that, according to Lemma 9.2.6 $\mathbf{cmp}(\mathbf{W}_y) \leq 2,\ y \in S$ and according to Lemma 9.2.5 $\mathbf{cmp}(\mathbf{G}_B) \leq 2$. We distinguish two cases.

Case 2.1. $Y = \{y_1, y_2\}$ where y_2 is light. Then let $\mathbf{G}_1 = (G[\{y_1, y_2\}], \{y_1, y_2\}, \{y_2\})$ and $\mathbf{G}_2 = (G[\{y_1, y_2\}], \{y_2\}, \{y_1\})$. Clearly $\mathbf{cmp}(\mathbf{G}_1) = 2$ and $\mathbf{cmp}(\mathbf{G}_2) = 1$. Therefore, if $\mathbf{G}' = \mathbf{glue}(\mathbf{G}_B, \mathbf{G}_1, \mathbf{W}_{y_2}, \mathbf{G}_2, \mathbf{W}_{y_1})$, then, from Lemma 9.1.2, $\mathbf{cmp}(\mathbf{G}') \leq 2$.

Case 2.2. $Y = \{y_1\}$. Let $\mathbf{G}' = \mathbf{glue}(\mathbf{G}_B, \mathbf{W}_{y_1})$, then, from Lemma 9.1.2, $\mathbf{cmp}(\mathbf{G}') \leq 2$.

In both cases, if $X = \{x\}$, then we set $\mathbf{G} = \mathbf{glue}(\mathbf{W}_x, \mathbf{G}')$ while if $X = \emptyset$ we set $\mathbf{G} = \mathbf{G}'$. In any case, we observe that, from Lemma 9.1.2, $\mathbf{cmp}(\mathbf{G}) \leq 2$. From Lemma 9.1.1, $\mathbf{cmp}(G) \leq 2$ contradicting to the first condition of Lemma 9.2.3.

As in both cases we reach a contradiction, G must contain at least one central cut-vertex.

4. Let C be the set of all central cut-vertices of G. For each $c \in C$, let $\mathcal{X}_c$ be the subset of $\mathcal{C}_G^{(v)}$ that contains all its members that are not fans. Clearly, $\mathcal{X}_c$ contains exactly two elements. Notice that none of the vertices in $C \setminus \{c\}$ belongs in the double rooted graphs in $\mathcal{C}_G^{(v)} \setminus \mathcal{X}_c$. Indeed, if this is the case for some vertex $y \in C \setminus \{c\}$, then the member of $\mathcal{C}_G^{(y)}$ that avoids c would be a subgraph of some member of $\mathcal{C}_G^{(v)} \setminus \mathcal{X}_c$, and this would imply that some fan would contain as a contraction some double rooted graph that is not a fan. We conclude that, for each $c \in C$ there is a partition $p(c) = (A_c, B_c)$ of $C \setminus \{c\}$ such that all members of A_c are vertices of one of the members of $\mathcal{X}_c$ and all members of B_c are vertices of the other.

We say that a vertex $c \in C$ is *extremal* if $p(c) = \{\emptyset, C \setminus \{c\}\}$

We claim that for any three vertices $\{x, y, z\}$ of C, there is one, say y of them such that x and z belong in different sets of $p(y)$. Indeed, if this is not the case, then one of the following would happen: Either there is a vertex $w \in C$ such that x, y, z belong to different elements of $\mathcal{C}_G^{(w)}$, a contradiction to the first statement of this lemma, or x, y, and z belong to the same block of G, a contradiction to the second statement of this lemma.

By the claim above, there is a path P containing all central cut-vertices in C. We can assume that this path is of minimum length, which permits us to further assume that its endpoints are extremal vertices of C. Moreover, heavy cut-vertices in $V(P)$ are members of C. Let $c_1, \ldots, c_{r+1}$ be the central cut-vertices ordered as they appear in P. As, for every $i \in \{1, \ldots, r\}$ there is a block B_i containing the central cut-vertices c_i and c_{i+1} we end up

with the two orderings required in the forth statement of the lemma. □

Let G be a graph in $\mathcal{Q}$. Suppose also that $c_1, \ldots, c_{r+1}$ and $B_1, B_2, \ldots, B_r$ are as in Lemma 9.2.7.4. We define the *extremal blocks* of G as follows:

- If $r > 0$, then among all blocks that contain c_1 as a cut-vertex let B_0 be the one such that $C_G(c_1, B_0)$, doubly rooted at c_1, is not a fan, does not contain any edge of the central blocks of G and does not contain c_{r+1}. Symmetrically, among all blocks that contain c_{r+1} as a cut-vertex let B_{r+1} be the one such that $C_G(c_{r+1}, B_{r+1})$ doubly rooted at c_{r+1} is not a fan, does not contain any edge of the central blocks of G and does not contain c_1.

- If $r = 0$, then let B_0 and B_1 be the two blocks with the property that for $i \in \{0, 1\}$, $C_G(c_1, B_i)$, doubly rooted at c_1, is not a fan.

We call B_0 and B_{r+1} *left* and *right* extremal block of G respectively. We also call the blocks of G that are either central or extremal *spine* blocks of G. Let $A(G)$ be the set of cut-vertices of the graphs $B_0, B_1, B_2, \ldots, B_r$, B_{r+1}. We partition $A(G)$ into three sets A_1, A_2 and A_3 as follows:

- $A_1 = \{c_1, \ldots, c_{r+1}\}$ (i.e., all central vertices).

- A_2 contains all vertices of $A(G)$ that belong to central blocks and are not central vertices.

- A_3 contains all vertices of $A(G)$ that belong to extremal blocks and are not central cut-vertices.

Moreover, we further partition A_3 to two sets: $A_3^{(0)} = A_3 \cap V(B_0)$ and $A_3^{(r+1)} = A_3 \cap V(B_{r+1})$ (see Figure 9.11 for an example).

Let $G \in \mathcal{Q}$ and $v \in A(G)$. We denote by $\mathcal{R}_G^{(v)}$ the set of the doubly rooted graphs in $\mathcal{C}_G^{(v)}$ that do not contain any of the edges of the spine blocks of G.

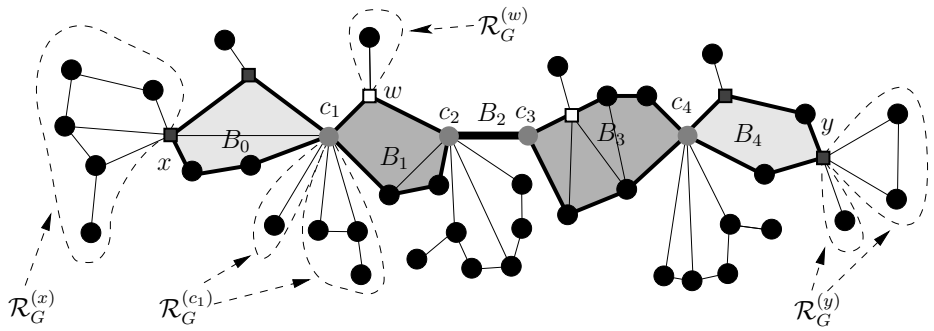


Figure 9.11: A graph G and the blocks $B_0, B_1, \ldots, B_3$, and B_4. The cut-vertices in $A_1 = \{c_1, \ldots, c_4\}$ are the grey circular vertices, the vertices in A_2 are the white square vertices and the vertices in A_3 are the dark square vertices.

LEMMA 9.2.8. Let $G \in \mathcal{Q}$ and $v \in A(G)$. The following hold:

a) Each doubly rooted graph in $\mathcal{R}_G^{(v)}$ is a fan.

b) All vertices in $v \in A_2$ are light, i.e., for each $v \in A_2$ $\mathcal{R}_G^{(v)}$ contains exactly one graph that is a hair block of G.

Proof. a) Let $v \in A(G)$. We distinguish two cases:

Case 1: $v \in A_1$. If there exists a double rooted graph in $\mathcal{R}_G^{(v)}$ that is not a fan, v will have spine-degree greater than 3, a contradiction to the first property of Lemma 9.2.7.

Case 2: $v \in A_2 \cup A_3$. If there exists a double rooted graph in $\mathcal{R}_G^{(v)}$ that is not a fan, v will have spine-degree greater than 2, therefore v must be central, a contradiction.

b) Let $v \in A_2$, and suppose that $\mathcal{R}_G^{(v)}$ can be contracted to two edges with v as their unique common endpoint, or to a triangle. As v belongs to

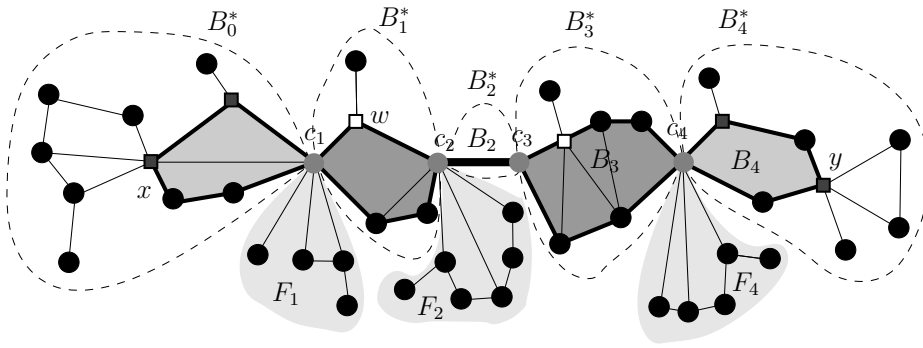


Figure 9.12: A graph G, the extended extremal blocks B_0^* and B_4^* the extended central blocks B_1^*, B_2^*, and B_3^* and the rooted graphs F_1, F_2 and F_4 (F_3 is the graph consisting only of the vertex c_3, doubly rooted on c_3).

a central block, G can be contracted to a graph in $\mathcal{O}_5$, a contradiction to the third property of Lemma 9.2.3. □

9.2.5 Directional obstructions

Let $G \in \mathcal{Q}$ and let $B_0, B_1, \ldots, B_r, B_{r+1}$ be the spine blocks of G. Notice first that, from Lemma 9.2.4.6, for every $i \in \{1, \ldots, r\}$, $|A_2 \cap V(B_i)| \leq 1$. Also, from Lemma 9.2.4.6, if $A_2 \cap V(B_i) = \{v\}$, then v is a light cut-vertex. For $i \in \{1, \ldots, r\}$, we define the rooted graphs $\mathbf{B}_i^*$ as follows: If $A_2 \cap V(B_i) = \{v\}$, then B_i^* is the union of B_i and the underlying graph of the unique rooted graph in $\mathcal{R}_G^{(v)}$ (this rooted graph is unique and its underlying graph is a hair block of G from the second statement of Lemma 9.2.8). If $A_2 \cap V(B_i) = \emptyset$, then B_i^* is B_i. We finally define the rooted graph $\mathbf{B}_i^* = (B_i^*, \{c_i\}, \{c_{i+1}\})$ for $i \in \{1, \ldots, r\}$.

We also define B_0^* as follows: Consider the unique graph B_0 in $\mathcal{C}_G^{(c_1)}$ that, when doubly rooted at c_1, is not a fan, does not contain any edge of the central blocks of G and does not contain c_{r+1}. Then $\mathbf{B}_0^* = (B_0, \emptyset, \{c_1\})$.

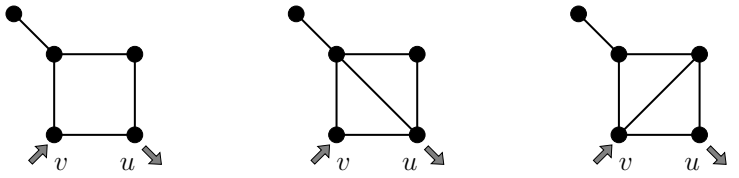


Figure 9.13: The set of rooted graphs $\mathcal{L}$ containing three rooted graphs each of the form $(G, \{v\}, \{u\})$.

Analogously, we define B^*_{r+1} by considering the graph B_{r+1} of the unique rooted graph in $\mathcal{R}_G^{(c_{r+1})}$ that, when doubly rooted at c_{r+1}, is not a fan, does not contain any edge of the central blocks of G and does not contain c_1. Then $\mathbf{B}^*_{r+1} = (B_{r+1}, \{c_{r+1}\}, \emptyset)$. Finally, we define for each $i \in \{1, \ldots, r+1\}$ the graph F_i that is the union of all the graphs of the rooted graphs in $\mathcal{R}_G^{(c_i)}$ that are fans (when performing the union, the vertex c_i stays the same), and in the case where $\mathcal{R}_G^{(c_i)}$ is empty, then F_i is the trivial graph $(\{c_i\}, \emptyset)$. We set $\mathbf{F}_i = (F_i, \{c_i\}, \{c_i\})$ $i \in \{1, \ldots, r+1\}$ and we call the rooted graphs $\mathbf{F}_1, \ldots, \mathbf{F}_{r+1}$ *extended fans* of G. We call $\mathbf{B}^*_0, \mathbf{B}^*_1, \ldots, \mathbf{B}^*_r, \mathbf{B}^*_{r+1}$ the *extended blocks* of the graph $G \in \mathcal{Q}$ and we – naturally – distinguish them in *central* and *extremal* (*left* or *right*), depending on the type of the blocks that contains them (see Figure 9.12 for an example). Notice that

$$\{E(B^*_0), E(F_1), E(B^*_1), E(F_2), \ldots, E(F_r), E(B^*_r), E(F_{r+1}), E(B^*_{r+1})\}$$

is a partition of the edges of G.

Lemma 9.2.9. Let $G \in \mathcal{Q}$ and let $\mathbf{B}^*_1, \ldots, \mathbf{B}^*_r$ be the central blocks of G. None of the rooted graphs in the set $\mathcal{L}$ of Figure 9.13 is a contraction of $\mathbf{B}^*_i$ if and only if $\mathbf{cmp}(\mathbf{B}^*_i) \leq 2$.

Proof. Clearly, for every graph $H \in \mathcal{L}$, $\mathbf{cmp}(H, \{v\}, \{u\}) = 3$, therefore if $\mathbf{B}^*_i$ can be contracted to a graph in $\mathcal{L}$, according to Lemma 9.1.4,

$\mathbf{cmp}(\mathbf{B}_i^*) \geq 3$.

Let now $\mathbf{B}_i^*$ be an central extended block of G that cannot be contracted to a graph in $\mathcal{L}$. If B_i does not contain some light cut-vertex, we define $\mathbf{G}_{B_i} = (B_i, \{c_i\}, \{c_{i+1}\})$. Notice that $\mathbf{B}_i^* = \mathbf{G}_{B_i}$ and, as from Lemma 9.2.5 $\mathbf{cmp}(\mathbf{G}_{B_i}) \leq 2$, we are done.

In the remaining case, where B_i contains a light cut-vertex, say c, observe that c cannot be adjacent via an outer edge to c_i, or else B_i^* could be contracted to a graph in $\mathcal{L}$. Therefore, according to Lemma 9.2.4.7, c is connected via an haploid edge with c_{i+1}. Notice that $\mathbf{G}_{B_i} = (B_i, \{c_i\}, \{c_{i+1}, c\})$. According to Lemma 9.2.5, $\mathbf{cmp}(\mathbf{G}_{B_i}) \leq 2$ and, according to Lemma 9.2.8, $\mathcal{R}^{(c)}$ contains only a hair block, say $(H, \{c\}, \{c\})$. Clearly $\mathbf{cmp}(H, \{c\}, \{c\}) = 2$. Let $\mathbf{G}_1 = (G[\{c, c_{i+1}\}], \{c, c_{i+1}\}, \{c\})$, $\mathbf{G}_2 = (G[\{c, c_{i+1}\}], \{c\}, \{c_{i+1}\})$, and $\mathbf{G} = \mathbf{glue}(\mathbf{G}_{B_i}, \mathbf{G}_1, (H, \{c\}, \{c\}), \mathbf{G}_2)$. From Lemma 9.1.2, $\mathbf{cmp}(\mathbf{G}) \leq 2$ and the lemma follows as $\mathbf{G} = \mathbf{B}_i^*$. □

Lemma 9.2.10. If $\mathbf{B}_i^*$ is one of the central extended blocks of a graph $G \in \mathcal{Q}$, then either $\mathbf{cmp}(\mathbf{B}_i^*) \leq 2$ or $\mathbf{cmp}(\mathbf{rev}(\mathbf{B}_i^*)) \leq 2$.

Proof. Proceeding towards a contradiction, from Lemma 9.2.9, both $\mathbf{B}_i^*$ and $\mathbf{rev}(\mathbf{B}_i^*)$ must contain one of the rooted graphs in Figure 9.13 as a contraction. It is easy to verify that, in this case, either B_i^* contains at least four cut-vertices, which contradicts to Lemma 9.2.4.9 or G can be contracted to either a graph in $\mathcal{O}_8$ (if the two roots are adjacent) or a graph in O_6 (if the two roots are not adjacent), a contradiction to Lemma 9.2.3.3. □

Now we give labels to central extended blocks. These labels are intended to indicate the way these blocks can be searched.

Definition 9.2.3. Let $G \in \mathcal{Q}$ and let $\mathbf{B}_i^*$ be one of the central extended blocks of G.

- If $\mathbf{rev}(\mathbf{B}_i^*)$ can be contracted to a graph in $\mathcal{L}$ (Figure 9.13), then we assign to $\mathbf{B}_i^*$ the label $\leftarrow$.

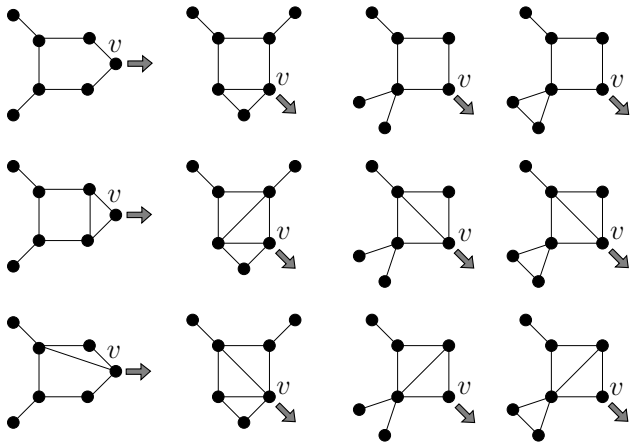


Figure 9.14: The set of rooted graphs $\mathcal{B}$ containing 12 rooted graphs each of the form $(G, \emptyset, \{v\})$.

- If $\mathbf{B}_i^*$ can be contracted to a graph in $\mathcal{L}$ (Figure 9.13), then we assign to $\mathbf{B}_i^*$ the label $\rightarrow$.
- If both $\mathbf{B}_i^*$ and $\mathbf{rev}(\mathbf{B}_i^*)$ can be contracted to a graph in $\mathcal{L}$ (Figure 9.13), then we assign to $\mathbf{B}_i^*$ the label $\leftrightarrow$.

Lemma 9.2.11. Let $G \in \mathcal{Q}$ and let $\mathbf{B}_0^*$ be the extended left extremal block of G. None of the rooted graphs in the set $\mathcal{B}$ in Figure 9.14 is a contraction of $\mathbf{B}_0^*$ if and only if $\mathbf{cmp}(\mathbf{B}_0^*) \leq 2$.

Proof. Clearly, for every graph $H \in \mathcal{B}$, $\mathbf{cmp}(H, \emptyset, \{u\}) = 3$. Therefore, if $\mathbf{B}_0^*$ can be contracted to a graph in $\mathcal{B}$, according to Lemma 9.1.4, $\mathbf{cmp}(\mathbf{B}_0^*) \geq 3$.

Suppose now that $\mathbf{B}_0^*$ cannot be contracted to a graph in $\mathcal{B}$. We distinguish three cases according to the number of cut-vertices in B_0 (recall that, from Lemma 9.2.4.4, B_0 can have at most 3 cut-vertices).

Case 1: B_0 contains only one cut-vertex, which is c_1. Then, $\mathbf{B}_0^* = \mathbf{G}_{B_0}$ and the result follows because of Lemma 9.2.5.

Case 2: B_0 contains two cut-vertices, c_1 and c. If B_0 has not a chord or it has a chord and c and c_1 are incident to two different haploid faces of B_0, then we can assume that $\mathbf{G}_{B_0} = (B_0, \{c\}, \{c_1\})$ and, from Lemma 9.2.5, $\mathbf{cmp}(\mathbf{G}_{B_0}) \leq 2$. According to Lemma 9.2.8.a, $\mathcal{R}^{(c)}$ is a fan, say $(F, \{c\}, \{c\})$ and from Lemma 9.2.6, $\mathbf{cmp}(F, \{c\}, \{c\}) \leq 2$. Let $\mathbf{G} = \mathbf{glue}((F, \{c\}, \{c\}), \mathbf{G}_{B_0})$. From Lemma 9.1.2, $\mathbf{cmp}(\mathbf{G}) \leq 2$. Combining the fact that $\mathbf{G} = (B_0^*, \{c\}, \{c_1\})$ with Lemma 9.1.1, $\mathbf{cmp}(\mathbf{B}_0^*) \leq 2$. In the remaining case c and c_1 are adjacent and c is light. Then $\mathbf{G}_{B_0} = (B_0, \emptyset, \{c, c_1\})$ and, from Lemma 9.2.5, $\mathbf{cmp}(\mathbf{G}_{B_0}) \leq 2$. According to Lemma 9.2.8.b, $\mathcal{R}^{(c)}$ is a hair, say $(H, \{c\}, \{c\})$. Let $\mathbf{G}_1 = (G[\{c, c_1\}], \{c, c_1\}, \{c\})$, $\mathbf{G}_2 = (G[\{c, c_1\}], \{c\}, \{c_1\})$ and $\mathbf{G} = \mathbf{glue}(\mathbf{G}_{B_0}, \mathbf{G}_1, (H, \{c\}, \{c\}), \mathbf{G}_2)$. Notice that $\mathbf{G} = (B_0^*, \emptyset, \{c_1\}) = \mathbf{B}_0^*$. From Lemma 9.1.2, we obtain that $\mathbf{cmp}(\mathbf{G}) \leq 2$, therefore $\mathbf{cmp}(\mathbf{B}_0^*) \leq 2$.

Case 3: B_0 contains three cut-vertices, c_1, c and x. We first examine the case where there is a partition $\{X, Y\}$ of $\{c_1, c, x\}$ such that $|Y| = 2$, $c_1 \in Y$, the cut-vertex in $Y \setminus \{c_1\}$ is light, and Y is an edge of B_0 that, in case B_0 is a chord, is haploid. In this case we claim that $\mathbf{cmp}(\mathbf{B}_0^*) \leq 2$. Indeed, we may assume that c be the light cut-vertex of $Y \setminus \{c_1\}$, thus $\mathbf{G}_{B_0} = (B_0, \{x\}, \{c, c_1\})$. According to Lemma 9.2.5, $\mathbf{cmp}(\mathbf{G}_{B_0}) \leq 2$. From Lemma 9.2.8.a, $\mathcal{R}^{(x)}$ is a fan, say $(F, \{x\}, \{x\})$ and, from Lemma 9.2.8.b, $\mathcal{R}^{(c)}$ contains only a hair block, say $(H, \{c\}, \{c\})$. Let $\mathbf{G}_1 = (G[\{c, c_1\}], \{c, c_1\}, \{c\})$, $\mathbf{G}_2 = (G[\{c, c_1\}], \{c\}, \{c_1\})$ and $\mathbf{G} = \mathbf{glue}((F, \{x\}, \{x\}), \mathbf{G}_{B_0}, \mathbf{G}_1, (H, \{c\}, \{c\}), \mathbf{G}_2)$. Notice that $\mathbf{G} = (B_0^*, \{x\}, \{c_1\})$ and. From Lemma 9.1.2 $\mathbf{cmp}(\mathbf{G}) \leq 2$. Now, Lemma 9.1.1 implies that $\mathbf{cmp}(\mathbf{B}_0^*) \leq 2$ and the claim holds.

In the remaining cases, the following may happen:

1. None of x and c is adjacent to c_1 via an edge that, in case B_0 has a chord, is haploid. In this case $\mathbf{B}_0^*$ can be contracted to the rooted graphs of the first column in Figure 9.14.

2. Both c_1 and x are light and only one of them, say x, is adjacent to c_1. In this case B_0 has a chord and either the edge $\{x, c_1\}$ is not haploid or $\{x, c_1\}$ is haploid and belongs in the same haploid face with c. In the first case, $\mathbf{B}_0^*$ can be contracted to the second rooted graph of the second column in Figure 9.14 and in the second case $\mathbf{B}_0^*$ can be contracted to the first and the third rooted graph of the second column in Figure 9.14.

3. c_1 has only one, say x, heavy neighbour in $\{c, x\}$ such that, in case B_0 has a chord, the edge $\{c_1, x\}$ is haploid. In this case $\mathbf{B}_0^*$ can be contracted to the rooted graphs of the third and the fourth column in Figure 9.14. □

Lemma 9.2.12. Let $G \in \mathcal{Q}$ and let $\mathbf{B}_{r+1}^*$ be the extended right extremal block of G. None of the rooted graphs in the set $\mathcal{C}$ in Figure 9.15 is a contraction of $\mathbf{B}_{r+1}^*$ if and only if $\mathbf{cmp}(\mathbf{B}_{r+1}^*) \leq 2$.

Proof. Clearly, for every graph $H \in \mathcal{C}$, $\mathbf{cmp}(H, \{u\}, \emptyset) = 3$, therefore if $\mathbf{B}_{r+1}^*$ can be contracted to a graph in $\mathcal{C}$, according to Lemma 9.1.4, $\mathbf{cmp}(\mathbf{B}_{r+1}^*) \geq 3$.

Suppose now that $\mathbf{B}_{r+1}^*$ cannot be contracted to a graph in $\mathcal{C}$. We distinguish three cases according to the number of cut-vertices in B_{r+1} (recall that, from Lemma 9.2.4.4, B_{r+1} can have at most 3 cut-vertices).

Case 1: B_{r+1} contains only one cut-vertex, which of course is c_1. Notice that $\mathbf{B}_{r+1}^* = \mathbf{G}_{B_{r+1}}$ and the result follows because of Lemma 9.2.5.

Case 2: If B_{r+1} has not a chord or it has a chord and c and c_1 are incident to two different haploid faces of B_{r+1} then we can assume that $\mathbf{G}_{B_{r+1}} = (B_{r+1}, \{c_{r+1}\}, \{c\})$ and from Lemma 9.2.5, $\mathbf{cmp}(\mathbf{G}_{B_{r+1}}) \leq 2$. In any other case, c_{r+1} and c are on the boundary of the same haploid face of

B_{r+1} and none of them belongs in the boundary of the other. Then $\mathbf{B}^*_{r+1}$ can be contracted to some of the rooted graphs in the second column of Figure 9.15.

Case 3: B_{r+1} contains three cut-vertices, c_{r+1}, c and x. If c and x are not adjacent, then $\mathbf{B}^*_{r+1}$ can be contracted to some of the rooted graphs in the first column of Figure 9.15. Otherwise, one, say c, of them will be light and the edge $\{x, c\}$ should be haploid. Then we can assume that $\mathbf{G}_{B_{r+1}} = (B_{r+1}, \{c_{r+1}\}, \{x, c\})$ and according to Lemma 9.2.5, $\mathbf{cmp}(\mathbf{G}_{B_{r+1}}) \leq 2$. From Lemma 9.2.8, $\mathcal{R}^{(x)}$ is a fan, say $(F, \{x\}, \{x\})$ and $\mathcal{R}^{(c)}$ contains only a hair block, say $(H, \{c\}, \{c\})$. Let $\mathbf{G}_1 = (G[\{c, x\}], \{c, x\}, \{c\})$, $\mathbf{G}_2 = (G[\{c, x\}], \{c\}, \{x\})$ and $\mathbf{G} = \mathbf{glue}(\mathbf{G}_{B_{r+1}}, \mathbf{G}_1, (H, \{c\}, \{c\}), \mathbf{G}_2, (F, \{x\}, \{x\}))$. Notice that $\mathbf{G} = (B^*_{r+1}, \{c_{r+1}\}, \{x\})$ and, because of Lemma 9.1.2, $\mathbf{cmp}(\mathbf{G}) \leq 2$. Applying Lemma 9.1.1, we conclude that $\mathbf{cmp}(\mathbf{B}^*_{r+1}) \leq 2$. □

Lemma 9.2.13. Let $G \in \mathcal{Q}$ and let $\mathbf{B}^*_0$ and $\mathbf{B}^*_{r+1}$ be the two extremal extended blocks of G. It is never the case that $\mathbf{B}^*_0$ contains some graph in $\mathcal{B}$ and $\mathbf{rev}(\mathbf{B}^*_0)$ contains some rooted graph in $\mathcal{C}$. Also it is never the case that $\mathbf{B}^*_{r+1}$ contains some graph in $\mathcal{C}$ and $\mathbf{rev}(\mathbf{B}^*_{r+1})$ contains some rooted graph in $\mathcal{B}$.

Proof. Let $\mathbf{G}$ be a rooted graph in $K = \{\mathbf{rev}(\mathbf{B}^*_0), \mathbf{B}^*_{r+1}\}$. We distinguish the following cases, that apply for both rooted graphs in K:

Case 1: $\mathbf{G}$ can be contracted to a graph in the first column of Figure 9.15 and $\mathbf{rev}(\mathbf{G})$ to a graph in the first column of Figure 9.14. Notice that every cut-vertex of $\mathbf{G}$ cannot be connected with an outer edge and therefore $\mathbf{G}$ can be contracted to a graph in $\mathcal{O}_6$.

Case 2: $\mathbf{G}$ can be contracted to a graph in the first column of Figure 9.15 and $\mathbf{rev}(\mathbf{G})$ to a graph in the second column of Figure 9.14. Notice that

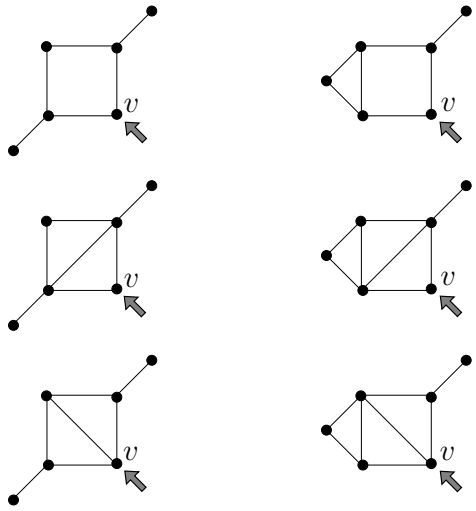


Figure 9.15: The set of rooted graphs $\mathcal{C}$ containing six rooted graphs each of the form $(G, \{v\}, \emptyset)$.

the two cut-vertices of **G**, that are other than the central cut-vertex, cannot be connected with an outer edge and therefore **G** can be contracted to graph in $\mathcal{O}_7$.

Case 3: **G** can be contracted to a graph in the first column of Figure 9.15 and **rev**(**G**) to a graph in the third or fourth column of Figure 9.14. Notice that the two cut-vertices of **G**, that are other than the central cut-vertex, cannot be connected with an outer edge and therefore **G** can be contracted to graph in $\mathcal{O}_8$.

Case 4: **G** can be contracted to a graph in the second column of Figure 9.15 and **rev**(**G**) to a graph in the first column of Figure 9.14. Notice that the two cut-vertices of **G**, that are other than the central cut-vertex, cannot be connected with an outer edge and therefore **G** can be contracted

to graph in $\mathcal{O}_7$.

Case 5: **G** can be contracted to a graph in the second column of Figure 9.15 and **rev**(**G**) to a graph in the second column of Figure 9.14. Notice that there must be an haploid face containing only the central cut-vertex, therefore **G** can be contracted either to the graph in $\mathcal{O}_2$ or to a graph in $\mathcal{O}_4$ (depending whether **G** can be contracted to the last graph in the second column of Figure 9.15 or not) .

Case 6: **G** can be contracted to a graph in the second column of Figure 9.15 and **rev**(**G**) to a graph in the third or fourth column of Figure 9.14. Notice that the light cut-vertex of **G** can not be connected via an haploid edge with the central cut-vertex, therefore **G** can be contracted to a graph in $\mathcal{O}_7$ either to a graph in $\mathcal{O}_9$ (depending whether the central cut-vertex is connected via haploid edge with a heavy cut-vertex or not). □

We will also give labels to the extremal extended blocks. These labels will indicate the way a search must start.

Definition 9.2.4. Let $G \in \mathcal{Q}$ and let $\mathbf{B}_0^*$, $\mathbf{B}_{r+1}^*$ be the extended left and right extremal blocks of G.

- If $\mathbf{B}_0^*$ contains some graph in $\mathcal{B}$ (Figure 9.14) then we assign to $\mathbf{B}_0^*$ the label $\leftarrow$.

- If **rev**$(\mathbf{B}_0^*)$ contains some graph in $\mathcal{C}$ (Figure 9.15) then we assign to $\mathbf{B}_0^*$ the label $\rightarrow$.

- If $\mathbf{B}_{r+1}^*$ contains some graph in $\mathcal{C}$ (Figure 9.15) then we assign to $\mathbf{B}_{r+1}^*$ the label $\leftarrow$.

- If **rev**$(\mathbf{B}_{r+1}^*)$ contains some graph in $\mathcal{B}$ (Figure 9.14) then we assign to $\mathbf{B}_{r+1}^*$ the label $\rightarrow$.

- If neither $\mathbf{B}_0^*$ contains some graph in $\mathcal{B}$ (Figure 9.14) nor $\mathbf{rev}(\mathbf{B}_0^*)$ contains some graph in $\mathcal{C}$(Figure 9.15) then we assign to $\mathbf{B}_0^*$ the label $\leftrightarrow$.

- If neither $\mathbf{B}_{r+1}^*$ contains some graph in $\mathcal{C}$ (Figure 9.15) nor $\mathbf{rev}(\mathbf{B}_{r+1}^*)$ contains some graph in $\mathcal{B}$ then we assign to $\mathbf{B}_{r+1}^*$ the label $\leftrightarrow$.

Lemma 9.2.14. Let $G \in \mathcal{Q}$ and let $\mathbf{B}_0^*, \mathbf{B}_1^*, \ldots, \mathbf{B}_r^*, \mathbf{B}_{r+1}^*$ be the extended blocks of G. It is not possible that one of these extended blocks is labeled with $\leftarrow$ and an other with $\rightarrow$.

Proof. We distinguish two cases according to the labelling of $\mathbf{B}_0^*$:

Case 1: Suppose that $\mathbf{B}_0^*$ is labeled $\leftarrow$ and that $\mathbf{B}_i^*$, for some $i \in \{1, \ldots, r+1\}$, is labeled $\rightarrow$. According to their respective labels, $(B_0^*, \emptyset, \{c_1\})$ can be contracted to a graph in $\mathcal{B}$ and, if $i \leq r$, $\mathbf{rev}(\mathbf{B}_i^*) = (B_i^*, \{c_{i+1}\}, \{c_i\})$ can be contracted to a graph in $\mathcal{L}$, otherwise $\mathbf{rev}(\mathbf{B}_{r+1}^*) = (B_{r+1}^*, \emptyset, \{c_{r+1}\})$ can be contracted to a rooted graph in $\mathcal{B}$. Notice that if $i \leq r$, $G[V(G) \setminus (V(B_1^*) \cup \cdots \cup V(B_{i-1}^*))], \emptyset, \{c_i\}\}$ can be contracted to a graph in the third and forth columns of $\mathcal{B}$. By further contracting all edges of $E(B_1^*) \cup \cdots \cup E(B_{i-1}^*)$ we obtain a graph in $\mathcal{O}_{11}$, a contradiction.

Case 2: Suppose now that $\mathbf{B}_0^*$ is labeled $\rightarrow$ and that $\mathbf{B}_i^*$, for some $i \in \{1, \ldots, r+1\}$, is labeled $\leftarrow$. According to their respective labels, $\mathbf{rev}(\mathbf{B}_0)^* = (B_0^*, \{c_1\}, \emptyset)$ can be contracted to a graph in $\mathcal{C}$ and, if $i \leq r$, $(B_i^*, \{c_i\}, \{c_{i+1}\})$ can be contracted to a graph in $\mathcal{L}$, otherwise $(B_{r+1}^*, \{c_{r+1}\}, \emptyset)$ can be contracted to a graph in $\mathcal{C}$. Notice that if $i \leq r$, $G[V(G) \setminus (V(B_1^*) \cup \cdots \cup V(B_{i-1}^*))], \{c_i\}, \emptyset\}$ can be be contracted to a graph in the first column of $\mathcal{C}$. By further contracting all edges of $E(B_1^*) \cup \cdots \cup E(B_{i-1}^*)$ we obtain a graph in $\mathcal{O}_{12}$, a contradiction. □

9.2.6 Putting things together

The last lemma we need to prove is the following:

LEMMA 9.2.15. $\mathcal{Q} = \emptyset$.

Proof. Suppose in contrary that $\mathcal{Q}$ contains some graph G. Let $\mathbf{B}_0^*, \mathbf{B}_1^*, \ldots, \mathbf{B}_r^*, \mathbf{B}_{r+1}^*$ and $\mathbf{F}_1, \ldots, \mathbf{F}_{r+1}$ be the extended blocks and fans of G, respectively. From Lemma 9.2.14, we can assume that the extended blocks of G are all labeled either $\rightarrow$ or $\leftrightarrow$ (if this is not the case, just reverse the ordering of the blocks). By the labelling of $\mathbf{B}_0^*$, none of the rooted graphs in the set $\mathcal{B}$ is a contraction of $\mathbf{B}_0^*$ therefore, from Lemma 9.2.11, $\mathbf{cmp}(\mathbf{B}_0^*) \leq 2$. Also as none of the rooted graphs $\mathbf{B}_i^*$, $i = 1, \ldots, r$, can be contracted to a graph in $\mathcal{L}$, from Lemma 9.2.9, it follows that $\mathbf{cmp}(\mathbf{B}_i^*) \leq 2$. We distinguish two cases according to the labelling of $\mathbf{B}_{r+1}^*$. If the labelling is $\leftrightarrow$, then $\mathbf{B}_{r+1}^*$ cannot be contracted to a graph in $\mathcal{C}$. If the labelling is $\rightarrow$, then $\mathbf{rev}(\mathbf{B}_{r+1}^*)$ can be contracted to a graph in $\mathcal{B}$ and, according to Lemma 9.2.14, $\mathbf{B}_{r+1}^*$ cannot be contracted to a graph in $\mathcal{C}$. Thus, in both cases, from Lemma 9.2.12, $\mathbf{cmp}(\mathbf{B}_{r+1}^*) \leq 2$. Notice that $(G, \emptyset, \emptyset) = \mathbf{glue}(\mathbf{B}_0^*, \mathbf{F}_1, \mathbf{B}_1^*, \ldots, \mathbf{F}_r, \mathbf{B}_r^*, \mathbf{F}_{r+1}, \mathbf{B}_{r+1}^*)$ and, from Lemma 9.1.2, $\mathbf{cmp}(G, \emptyset, \emptyset) \leq 2$. This implies that $\mathbf{cmp}(G) \leq 2$, a contradiction to the first property of Lemma 9.2.3. □

This completes the proof of Lemma 9.2.2 and show us that $\mathsf{obs}_{\leq_c}(\mathbf{cmms}, 2)$ contains exactly the graphs of $\mathcal{D}^1$. If someone look closely to the graphs of $\mathsf{obs}_{\leq_c}(\mathbf{cmms}, 2)$, it is not very difficult to reach the following conclusion.

COROLLARY 9.2.1. $\mathsf{obs}_{\leq_c}(\mathbf{cmms}, 2) = \mathsf{obs}_{\leq_c}(\mathbf{cms}, 2)$.

Proof. It is easy to check that for every $H \in \mathsf{obs}_{\leq_c}(\mathbf{cmms}, 2)$:

1. $\mathbf{cms}(H) \geq 3$,
2. for every proper contraction H' of H it holds that $\mathbf{cms}(H') \leq 2$,

therefore $\mathsf{obs}_{\leq_c}(\mathbf{cmms}, 2) \subseteq \mathsf{obs}_{\leq_c}(\mathbf{cms}, 2)$.

If there exists a graph $H \in \mathsf{obs}_{\leq_c}(\mathbf{cms}, 2) \setminus \mathsf{obs}_{\leq_c}(\mathbf{cmms}, 2)$, then $\mathbf{cms}(H) \geq 3$. Notice that the connected search number of a graph is always bounded from the monotone and connected search number, as a complete monotone and connected search strategy is obviously a complete connected search strategy, therefore $\mathbf{cmms}(H) \geq 3$, which means that there exists a graph $H' \in \mathsf{obs}_{\leq_c}(\mathbf{cmms}, 2)$ such that $H' \leq_c H$. Furthermore, since H' is a proper contraction of H, according to Lemma 9.1.5, $\mathbf{cms}(H) \geq \mathbf{cms}(H')$. As we have already stressed that $\mathbf{cms}(H') \geq 3$ we reach a contradiction to the minimality (with respect of $\leq_c$) of H. □

9.3 General Obstructions for cmms

As mentioned before, for $k > 2$, we have no guarantee that the set $\mathsf{obs}_{\leq_c}(\mathbf{cmms}, k)$ is a finite set. In this Section we prove that, when this set is finite, its size should be double exponential in k. Therefore, it seems hard to extend the results presented in this Chapter for $k \geq 3$ as, even if we somehow manage to prove that the obstruction set for a specific k is finite (see [99, 132] for a couple cases where a contraction obstruction set *is not finite*), then this set would contain more than $2^{2^{\Omega(k)}}$ graphs.

We will describe a procedure that generates, for each k, a set of at least $\frac{4}{3}(\frac{5}{2})^{3\cdot 2^{k-2}}$ non-isomorphic graphs that have connected, and monotone search number $k+1$ and are contraction-minimal with respect to this property. Hence, these $\frac{4}{3}(\frac{5}{2})^{3\cdot 2^{k-2}}$ graphs will belong to $\mathsf{obs}_{\leq_c}(\mathbf{cmms}, k)$.

Let us first define a set of rooted graphs which will produce these obstructions.

Definition 9.3.1. For every $k \geq 1$ we define the set of *obstruction-branches*, denoted by $\mathbf{Br}(k)$, as follows:

For $k = 1$: The set $\mathbf{Br}(1)$ consists of the five graphs of Figure 9.10 rooted at v.

For $k = l > 1$: The set $\mathbf{Br}(l)$ is constructed by choosing two *branches* of the set $\mathbf{Br}(l-1)$ and identify the two roots to a single vertex, say v. Then we add a new edge with v as an endpoint, say $\{u, v\}$, and we root this branch to u. We will refer to this edge as the *trunk* of the branch.

Let $f(k)$ be the number of branches of $\mathbf{Br}(k)$. Notice that $f(1) = 5$ and $f(k)$ is equal to the number of ways we can pick two branches of $\mathbf{Br}(k-1)$, with repetition. Therefore:

$$\begin{aligned}
f(k) &= \binom{f(k-1)+2-1}{2} = \binom{f(k-1)+1}{2} \\
&= \frac{f(k-1)^2 + f(k-1)}{2} \geq \frac{f(k-1)^2}{2} \\
&\geq \frac{\left(\frac{f(k-2)^2}{2}\right)^2}{2} = \frac{f(k-2)^{2^2}}{2^{2+1}} \geq \frac{f(k-3)^{2^3}}{2^{2^2+2+1}} \\
&\geq \cdots \geq \frac{f(1)^{2^{k-1}}}{2^{2^{k-2}+\cdots+2+1}} = \frac{5^{2^{k-1}}}{2^{2^{k-1}}} \\
&= 2\left(\frac{5}{2}\right)^{2^{k-1}}
\end{aligned}$$

Definition 9.3.2. Let $\mathcal{O}_{\mathbf{Br}}(k)$ be the set containing the graphs obtained by choosing three rooted branches of $\mathbf{Br}(k)$, with repetitions, and identify the three roots.

Notice that any two selections according to the definition above produce two non-isomorphic graphs.

We are going to prove that:

$$\mathcal{O}_{\mathbf{Br}}(k) \subseteq \mathsf{obs}_{\leq_c}(\mathbf{cmms}, k+1)$$

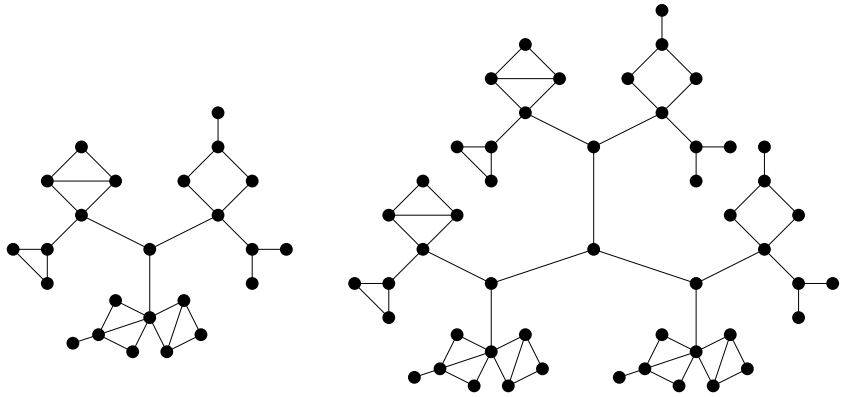


Figure 9.16: The left graph belongs to $\mathcal{O}_{\mathbf{Br}}(2)$ and the right to $\mathcal{O}_{\mathbf{Br}}(3)$.

Notice that:

$$\begin{aligned}
|\mathcal{O}_{\mathbf{Br}}(k)| &= \binom{f(k)+3-1}{3} = \binom{f(k)+2}{3} \\
&= \frac{(f(k)+2)(f(k)+1)f(k)}{6} \\
&= \frac{f(k)^3+3f(k)^2+2f(k)}{6} \\
&\geq \frac{f(k)^3}{6} \geq \frac{4}{3}\left(\frac{5}{2}\right)^{3\cdot 2^{k-1}}
\end{aligned}$$

Hence, the cardinality of $\mathsf{obs}_{\leq_c}(\mathbf{cmms}, k)$ is at least $\frac{4}{3}\left(\frac{5}{2}\right)^{3\cdot 2^{k-2}}$. In order to prove this we need the following lemmata.

Lemma 9.3.1. Let $B \in \mathbf{Br}(k)$ and let v be its root. There does not exist a complete connected and monotone search strategy for B – that uses k searchers – such that the first edge being cleaned is the trunk of B.

Proof. We are going to prove this by induction. We can easily check that for $k = 1$ the claim holds. Let $B \in \mathbf{Br}(k)$ and let v be its root and u the other endpoint of the trunk. Since we are forced to clean B, in a connected and monotone manner, with a search strategy, say $\mathcal{S}$, that first cleans $\{u, v\}$, a searcher must be placed in u during each step of $\mathcal{S}$, therefore we must clean a $(k-1)$-level branch using $k-1$ searchers that first clean the trunk of this branch, which contradicts the induction hypothesis. □

COROLLARY 9.3.1. Let $G \in \mathcal{O}_{\mathbf{Br}}(k)$, then $\mathbf{cmms}(G) > k + 1$.

Proof. Let $G \in \mathcal{O}_{\mathbf{Br}}(k)$. Notice that G consists of three k-level obstruction branches, say B_1, B_2 and B_3. If there exist a complete connected and monotone search strategy $\mathcal{S}$ that uses $k+1$ searchers, then from Lemma 9.3.1 $\mathcal{S}$ cannot start by placing searchers in the central vertex, i.e., the vertex where B_1, B_2 and B_3 are connected. Therefore, $\mathcal{S}$ starts by placing searchers in a vertex of B_1, B_2 or B_3 and consequently the first edge cleaned belongs to this branch. Notice that the first time that a searcher is placed on the central vertex the connectivity and monotonicity of $\mathcal{S}$ force us to clean a k-level branch with k searchers, which is impossible according to Lemma 9.3.1. □

LEMMA 9.3.2. Let $B \in \mathbf{Br}(k)$ and let v be its root.

a) There exist a complete connected and monotone search strategy for B – that uses $k+2$ searchers – such that in each step a searcher occupies v.

b) There exists a complete connected and monotone search strategy for B – that uses $k+1$ searchers – such that the first edge being cleaned is the trunk of B.

c) There exist a complete connected and monotone search strategy for B – that uses $k+1$ searchers – such that the last edge being cleaned is the trunk of B.

Proof. a) We are going to prove this by induction. We can easily check that for $k = 1$ the claim holds. Let $B \in \mathbf{Br}(k)$ and let v be its root and u the other endpoint of the trunk. We are going to describe a search strategy $\mathcal{S}$ with the properties needed. We place a searcher in v and as second searcher in u. According to the induction hypothesis for each one of the two $(k-1)$-level branches connected to u there exists a complete connected and monotone search strategy that uses $k+1$ searchers such that in each step a searcher occupies u, therefore, we can continue by cleaning one of these $(k-1)$-level branches and then clean the other.

b) We are going to prove this by induction. For $k = 1$ the claim is trivial. Let $B \in \mathbf{Br}(k)$ and let v be its root and u the other endpoint of the trunk. There are two $(k-1)$-level branches connected to u, say B_1 and B_2. The search strategy, say $\mathcal{S}$, with the properties needed is the following: we place a searcher in v and then slide him to u. According to the first claim of Lemma 9.3.2 there exists a complete connected and monotone search strategy $\mathcal{S}_1$ for B_1 that uses $k+1$ searchers such that in each step a searcher occupies u. By the induction hypothesis there exists a complete connected and monotone search strategy $\mathcal{S}_2$ for B_2 that uses k searchers such that the first edge cleaned is the trunk of B_2. Using these two search strategies we can start by cleaning B_1, keeping in all times a searcher in u, and then we can clean B_2.

c) We are going to prove this by induction. Notice that for $k = 1$ the claim holds. Let $B \in \mathbf{Br}(k)$ and let v be its root and u the other endpoint of the trunk. There are two $(k-1)$-level branches connected to u, say B_1 and B_2. According to the induction hypothesis there exist a complete connected and monotone search strategy $\mathcal{S}_1$ for B_1 that uses k searchers such that the last edge cleaned is the trunk of B_1. Moreover, according to the first claim of Lemma 9.3.2 there exists a complete connected and monotone search strategy $\mathcal{S}_2$ for B_2 that uses $k+1$ searchers such that in each step a searcher occupies u. Using these two search strategies we

can clean B, in a connected and monotone manner, as follows: We start by cleaning B_1 then we clean B_2, keeping in all times a searcher in u, and then we clean $\{u, v\}$. □

LEMMA 9.3.3. Let $G \in \mathcal{O}_{\mathbf{Br}}(k)$ and $B \in \mathbf{Br}(k)$ one of the three branches of G. If we contract an edge of B there exist a complete connected and monotone search strategy for B that uses $k+1$ searchers, such that in each step a searcher occupies v.

Proof. We are going to prove this by induction. It is easy to check that for $k = 1$ the claim is true. Let v be the root of B and u the other endpoint of the trunk, B_1 and B_2 the two $(k-1)$-level branches connected to u and $e \in E(B)$ the edge contracted. We distinguish to cases:

Case 1: $e \in E(B_1) \cup E(B_2)$. We can assume that e is an edge of B_1. We are going to describe a search strategy $\mathcal{S}$ for B with the properties needed. We place a searcher in v and a second searcher in u. From the induction hypothesis there exists a complete connected and monotone search strategy $\mathcal{S}_1$ for B_1 – that uses k searchers – such that in each step a searcher occupies u. Moreover, according to the second claim of Lemma 9.3.2 there exists a complete connected and monotone search strategy $\mathcal{S}_2$ for B_2 – that uses k searchers – such that the first edge cleaned is the trunk of B_2. Using these two search strategies we can start by cleaning B_1, keeping in all times a searcher in u, and then we can clean B_2. Notice that this search strategy uses $k+1$ searchers and during each step a searcher occupies v.

Case 2: $e = \{u, v\}$. According to the first property of Lemma 9.3.2, for each one of B_1 and B_2 there exists a complete connected and monotone search strategy – that uses $k+1$ searchers – such that in each step a searcher occupies v. Hence, we can clean B starting by cleaning B_1, keeping in all times a searcher in v, and then clean B_2. □

COROLLARY 9.3.2. If $G \in \mathcal{O}_{\mathbf{Br}}(k)$ and G' be a contraction of G, then $\mathbf{cmms}(G') = k + 1$.

Proof. It suffices to prove this claim for a single edge contraction.

Let $G \in \mathcal{O}_{\mathbf{Br}}(k)$, let B_1, B_2, and B_3 be the three k-level obstruction-branches of G connected to v, $e \in E(G)$ the edge contracted and G' the graph obtained from G after the contraction of e. We can assume that $e \in E(B_2)$. We are going to describe a complete connected and monotone search strategy $\mathcal{S}$ for G. From the third claim of Lemma 9.3.2 we know that there exist a complete connected and monotone search strategy $\mathcal{S}_1$ for B_1 – that uses $k+1$ searchers – such that the last edge cleaned is the trunk of B_1. From Lemma 9.3.3 we know that there exist a complete connected and monotone search strategy $\mathcal{S}_2$ for B_2 – that uses $k+1$ searchers – such that in each step a searcher occupies the root of B_2, in other words v. From the second claim of Lemma 9.3.2 we know that there exist a complete connected and monotone search strategy $\mathcal{S}_3$ for B_3 – that uses $k+1$ searchers – such that the first edge cleaned is the trunk of B_3. Therefore, we can clean G' starting by cleaning B_1 according to $\mathcal{S}_1$ (notice that the trunk of B_1 will be the last edge of $E(B_1)$ being cleaned), then clean B_2 according to $\mathcal{S}_2$, keeping in all times a searcher in v, and finish by cleaning B_3 according to $\mathcal{S}_3$. □

Combining Corollaries 9.3.1 and 9.3.2 we conclude that every graph in $\mathcal{O}_{\mathbf{Br}}(k)$ is a contraction obstruction for the graph class $\mathcal{G}[\mathbf{cmms}, k+1]$ and therefore $\mathcal{O}_{\mathbf{Br}}(k) \subseteq \mathsf{obs}_{\leq_c}(\mathbf{cmms}, k+1)$.

9.4 Conclusion

In this Chapter we showed that both $\mathsf{obs}_{\leq_c}(\mathbf{cmms}, 2)$ and $\mathsf{obs}_{\leq_c}(\mathbf{cms}, 2)$ are finite, and we gave a complete list of the graphs in these sets. Therefore, we managed to characterized these two graph classes, $\mathcal{G}[\mathbf{cmms}, 2]$ and $\mathcal{G}[\mathbf{cms}, 2]$, using forbidden contractions. Forbidden contractions characterizations are very rare (e.g., [99, 128, 132]), and finite one even rarer (e.g., [128]).

From the algorithmic point of view, we can use a cubic-time *contraction checking algorithm* for planar graphs (e.g., [46,47]) to obtain a cubic algorithm deciding whether a graph belongs to $\mathcal{G}[\mathbf{cms}, 2]$ (or $\mathcal{G}[\mathbf{cmms}, 2]$). An even faster approach will be to use Theorem 5.3.1, the fact that the class $\mathcal{C}$ of graphs not containing as contraction a graphs in $\mathcal{G}[\mathbf{cms}, 2]$ is MSO-definable, and the fact that for every graph G

$$\mathbf{tw}(G) \leq \mathbf{pw}(G) \leq \mathbf{ns}(G) \leq \mathbf{ms}(G) + 1 \leq \mathbf{cms}(G) + 1$$

(this follows from the definition of **tw** and **pw**, Theorem 8.4.1, Observation 8.1.1, and the definition of **ms** and **cms**). This approach yields the following linear-time algorithm:

Step 1: Use the algorithm of Theorem 4.1.2 and check whether the input graph G has $\mathbf{tw}(G) > k$. If so, return No and **stop**.

Step 2: If $\mathbf{tw}(G) \leq k$, run the algorithm of Theorem 5.3.1 and check if $G \in \mathcal{C}$.

As we have mentioned a number of times before, the contraction relation is not a well-quasi-ordering on $\mathcal{G}$, thus, we do not have a priori knowledge that the obstruction set of a contraction-closed graph class will be finite. Hence, the following question has great significance:

> *For what values of* $k \geq 2$ $\mathsf{obs}_{\leq_c}(\mathbf{cmms}, k)$ *and* $\mathsf{obs}_{\leq_c}(\mathbf{cms}, k)$ *are finite?*

Our "guts feeling" is that for some $k \geq 3$ (perhaps some small values) these sets are finite but not for all $k \geq 3$. We hope that this question would be answered sometime soon. On the other hand we do not believe that complete lists of the graphs of these sets are possible to be found, when they are finite, due to their – double exponential on k – size. At least not without the use of computer programs.

We have not given much information about the computational complexity of the problem of finding the search number of a graph. Let us define the following problems:

k-SEARCH	
Input:	A graph G, a positive integer k.
Question:	Is $\bullet(G) \leq k$?

where $\bullet \in \{\mathbf{es}, \mathbf{ns}, \mathbf{ms}, \mathbf{ces}, \mathbf{cns}, \mathbf{cms}\}$[3].

For the three basic types of searching, that is edge search, node search and mixed searched, this problem is NP-complete.

THEOREM 9.4.1 (Megiddo, Hakimi, Garey, Johnson, and Papadimitriou, 1983 [115])**.** *The k-SEARCH problem for edge search is* NP*-complete.*

If we consider the relation between the three search numbers **es**, **ns** and **ms** (see Observation 8.1.1) we can prove that:

THEOREM 9.4.2. *The k-SEARCH problem for node search is* NP*-hard.*

THEOREM 9.4.3. *The k-SEARCH problem for mixed search is* NP*-hard.*

In order to prove that these problems are in NP we have to prove the existence of a certificate of polynomial size that will certify that $\bullet(G) \leq k$. A search strategy will be perfect for this purpose. The problem is that, if we allow non-monotone search strategies, we may not have a polynomial number of moves in a strategy and, therefore, no polynomial certificate. As edge search, node search and mixed search are monotone we do not face this obstacle. But Theorem 8.3.1 shows us that connected edge search is not monotone. We believe that the same holds for connected node search and connected mixed search, thus, the following question not only remains unanswered, but also seems to be a hard one:

[3]Remeber that for a graph G $\mathbf{es}(G) = \mathbf{mes}(G)$, $\mathbf{ns}(G) = \mathbf{mns}(G)$, and $\mathbf{ms}(G) = \mathbf{mms}(G)$.

*Is the k-*Search *problem in* NP *for connected mixed search? If so, is it* NP*-complete?*

To show the difficulty of this question – at least at this moment – we pose another substantially easier question:

Is the 3-Search *problem or the* 4-Search *problem for connected mixed search in* NP*? If so, is it* NP*-complete?*

We also have to mention that we do not know the answer to these questions even for the case of connected and monotone mixed search.

Changing our perspective and looking at this problem from the parametrerized complexity point of view, we pose the following question:

*Is the k-*Search *problem parameterized by k in* FPT*?*

The answer in not known yet, as all existing techniques fail to provide as with one.

Appendices

APPENDIX A

SOME NICE FIGURES OF OBSTRUCTIONS!

Here, we present three obstruction sets, mentioned in Chapters 4 and 8. As their size is by far larger than the other sets presented in this thesis – except from those of Chapter 9) – we thought it would be better to move there depiction here.

We remind you the three theorems completely characterizing $\mathsf{obs}_{\leq_m}(\mathbf{lw}, 2)$, $\mathsf{obs}_{\leq_m}(\mathbf{ms}, 2)$, and $\mathsf{obs}_{\leq_m}(\mathbf{ns}, 3)$.

Theorem 4.1.4 (Thilikos, 2000 [120])**.** *The obstruction set* $\mathsf{obs}_{\leq_m}(\mathbf{lw}, 2)$ *consists of the 52 graphs shown in Figure A.1.*

Theorem 8.5.3 (Takahashi, Ueno, and Kajitani, [136])**.** *The obstruction set* $\mathsf{obs}_{\leq_m}(\mathbf{ms}, 2)$ *consists of the 36 graphs shown in Figure A.2.*

Theorem 8.5.4 (Kinnersley and Langston, [133])**.** *The obstruction set* $\mathsf{obs}_{\leq_m}(\mathbf{ns}, 3)$ *consists of the 110 graphs shown in Figures A.3, A.4, A.5 and A.6.*

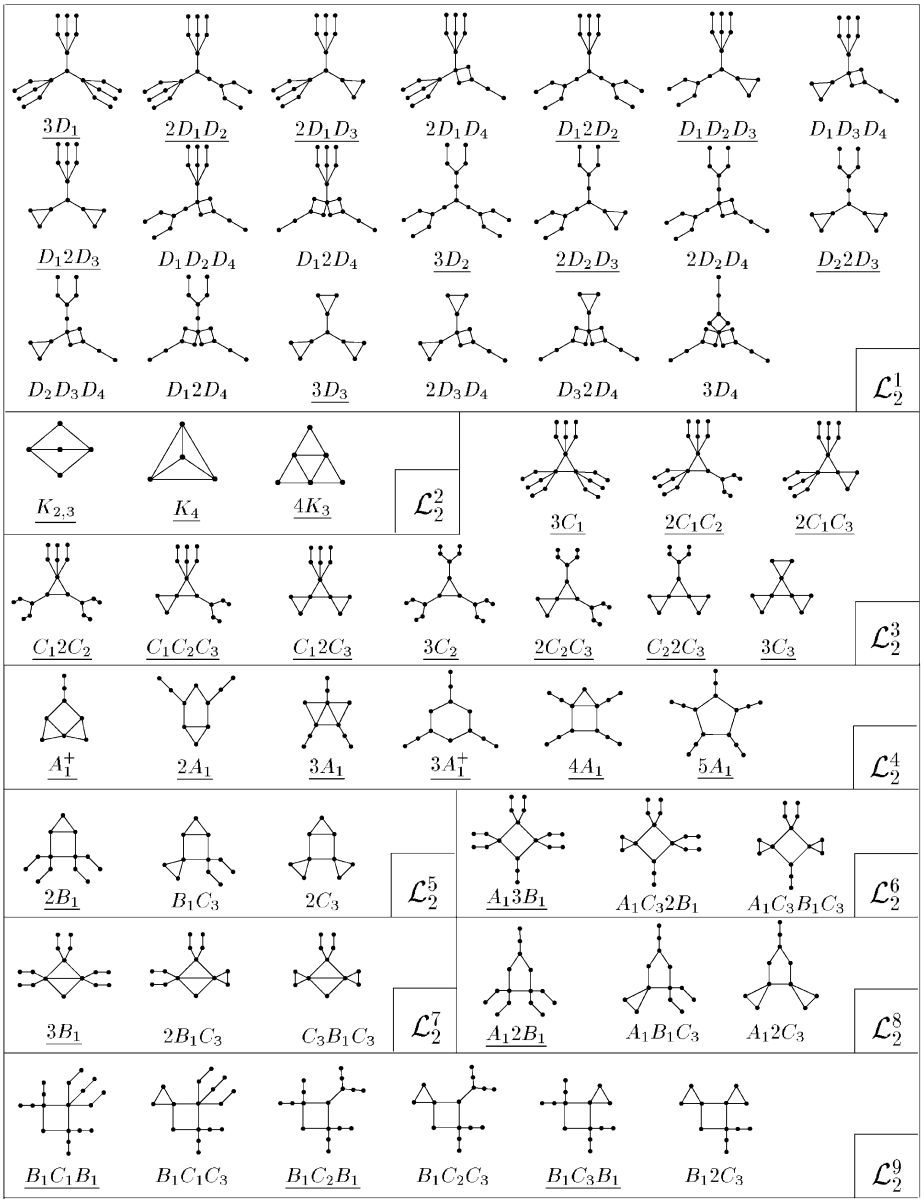


Figure A.1: The graphs of $\mathsf{obs}_{\leq \mathrm{m}}(\mathbf{lw}, 2)$.

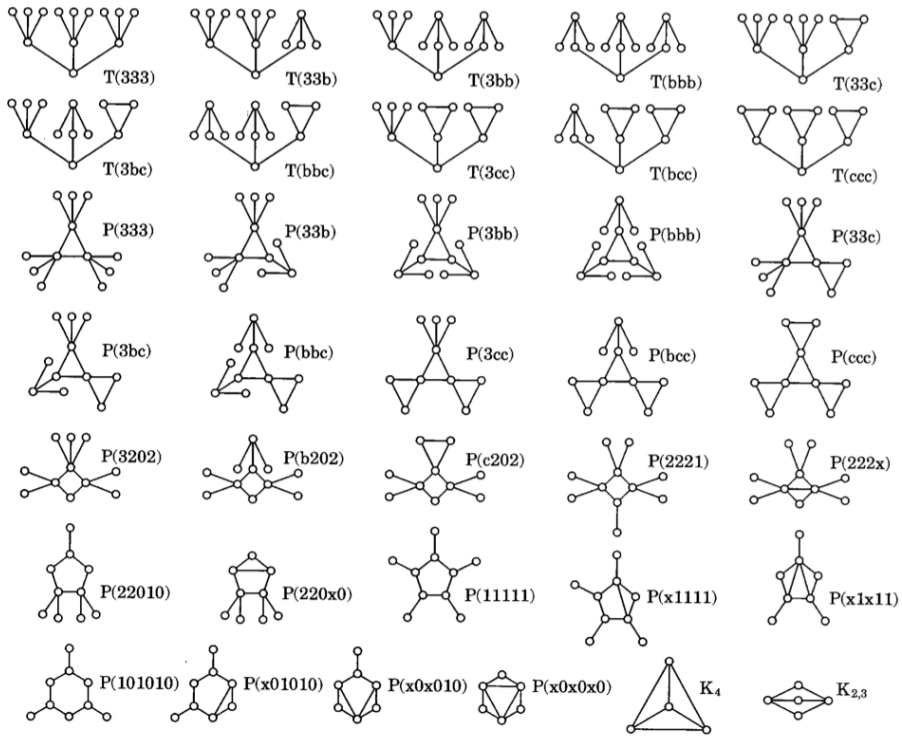


Figure A.2: The set $\mathsf{obs}_{\leq_{\mathrm{m}}}(\mathbf{ms}, 2)$.

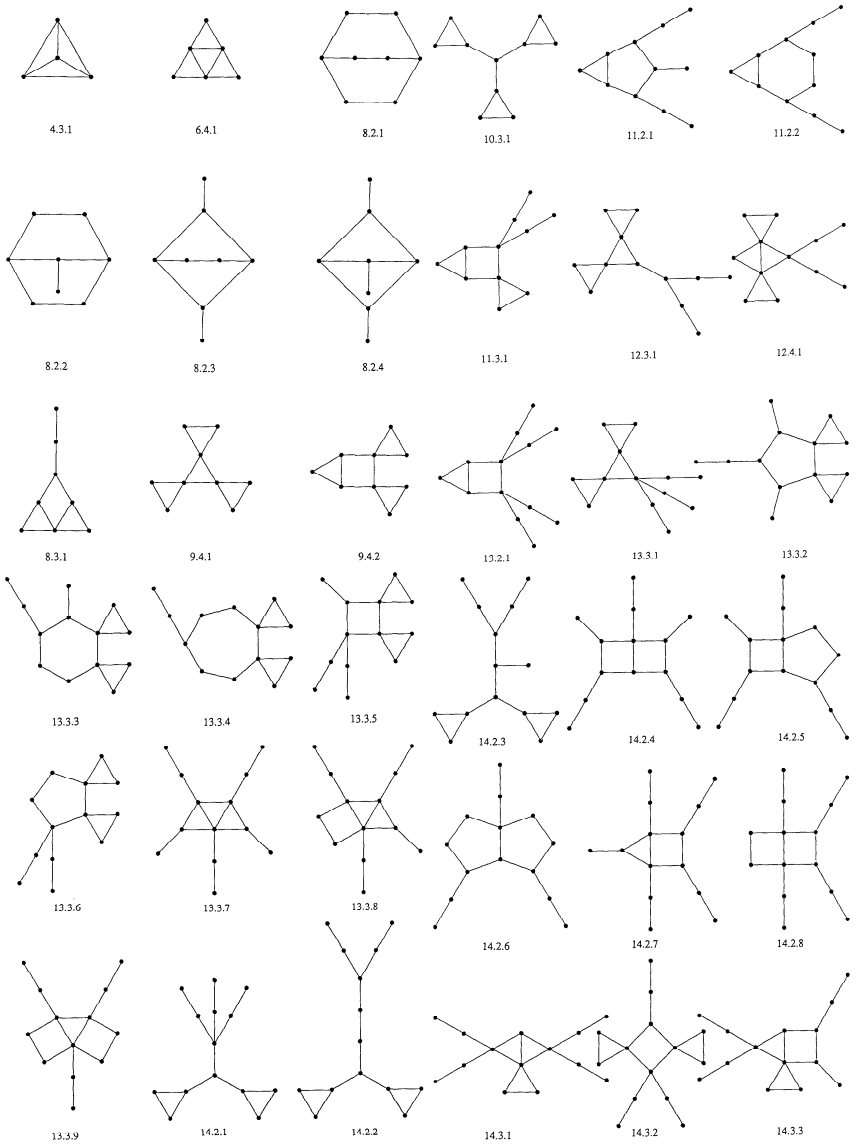


Figure A.3: The first part of the set $\mathrm{obs}_{\leq \mathrm{m}}(\mathbf{ns}, 3)$.

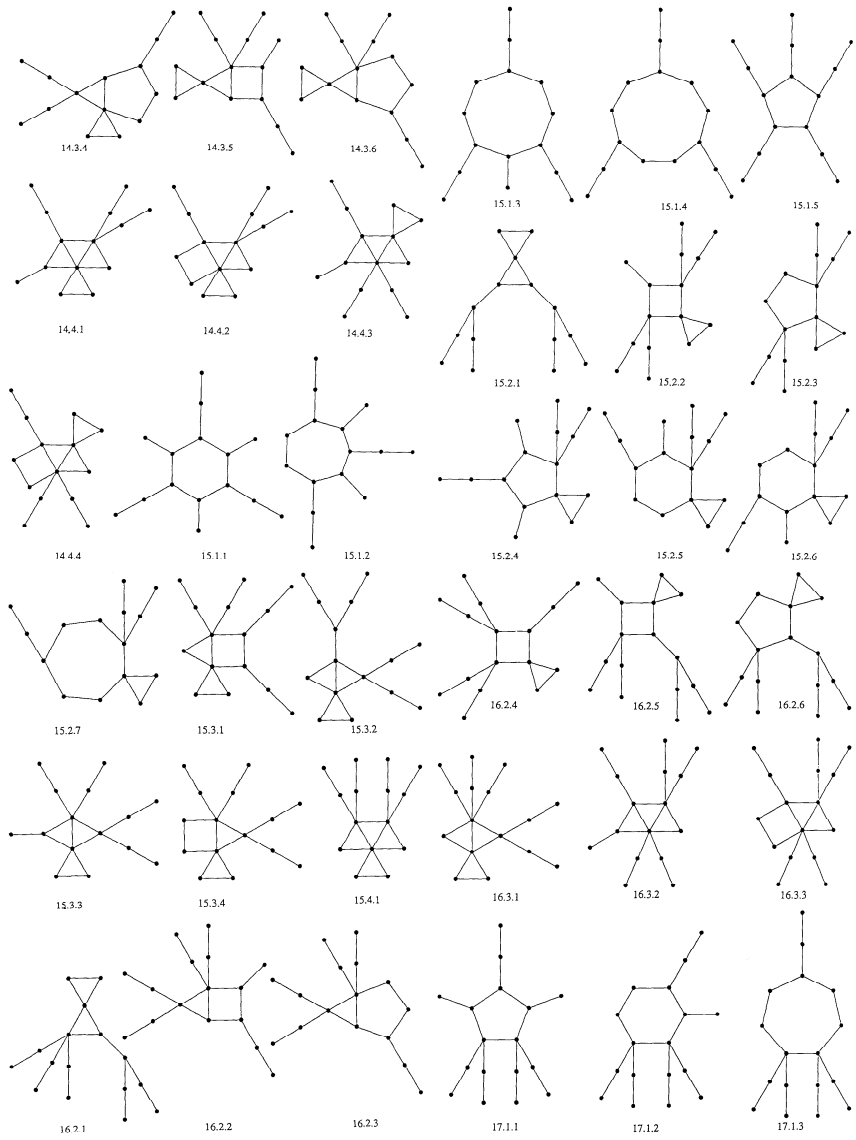


Figure A.4: The second part of the set $\mathsf{obs}_{\leq \mathrm{m}}(\mathbf{ns}, 3)$.

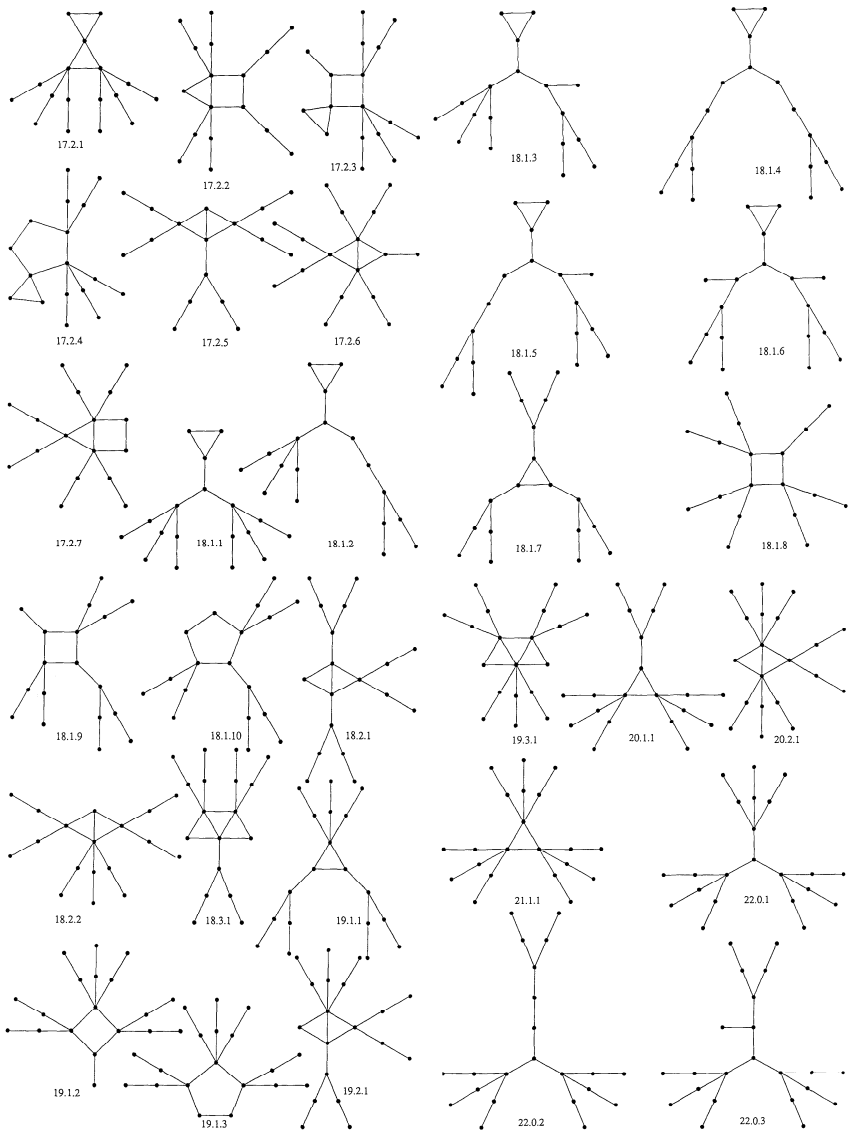


Figure A.5: The third part of the set $\mathrm{obs}_{\leq \mathrm{m}}(\mathbf{ns}, 3)$.

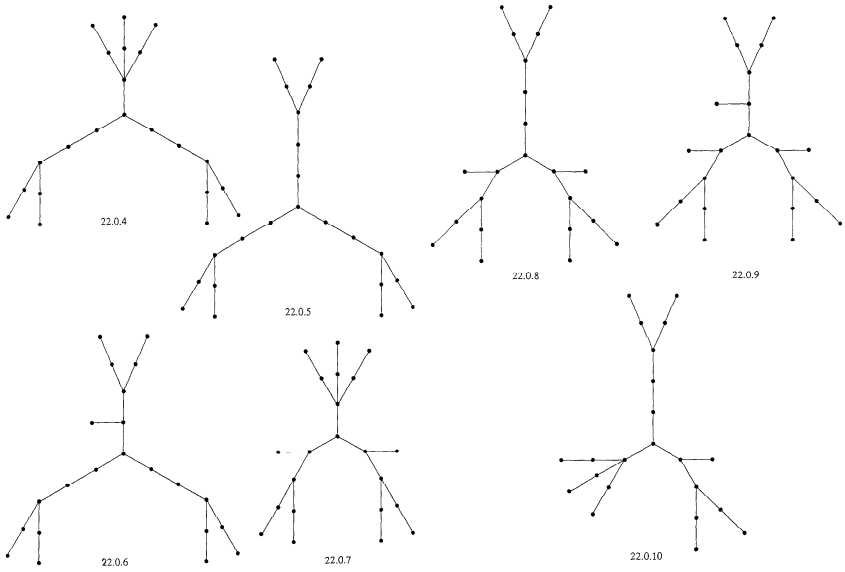


Figure A.6: The forth part of the set $\mathsf{obs}_{\leq \mathrm{m}}(\mathbf{ns}, 3)$.

BIBLIOGRAPHY

Thesis Papers

[1] Micah J Best, Arvind Gupta, Dimitrios M. Thilikos, and Dimitris Zoros, *Contraction Obstructions for Connected Graph Searching*, Discrete Applied Mathematics, Volume 209, pp. 27–47, 2016.

[2] Micah J Best, Arvind Gupta, Dimitrios M. Thilikos, and Dimitris Zoros, *Contraction Obstructions for Connected Graph Searching*, 9th International Colloquium on Graph Theory and Combinatorics (ICGT 2014), 2014.

[3] Dimitris Chatzidimitriou, Dimitrios M. Thilikos, and Dimitris Zoros, *Parameter Invariant, Minor-monotone Kernels*, Unpublished Manuscript, Submitted to: 12th International Symposium on Parameterized and Exact Computation (IPEC 2017), 2017.

[4] Archontia C. Giannopoulou, Iosif Salem, and Dimitris Zoros, *Effective Computation of Immersion Obstructions for Unions of Graph Classes*, Journal of Computer and System Sciences, Volume 80, Issue 1, pp. 207–216, 2014.

[5] Archontia C. Giannopoulou, Iosif Salem, and Dimitris Zoros, *Effective Computation of Immersion Obstructions for Unions of Graph Classes*, Scandinavian Workshop on Algorithm Theory (SWAT 2012), pp. 165-176, 2012.

[6] Menelaos I. Karavelas, Spyridon Maniatis, Dimitrios M. Thilikos, and Dimitris Zoros, *Geometric Extensions of Cutwidth in any Dimension*, 9th International Colloquium on Graph Theory and Combinatorics (ICGT 2014), 2014.

Papers Not Included In This Thesis

[7] Dimitris Chatzidimitriou, Archontia C. Giannopoulou, Spyridon Maniatis, Clément Requile, Dimitrios M. Thilikos, and Dimitris Zoros, *Fixed Parameter Algorithms for Completion Problems on Planar Graphs*, 41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016), 2016.

[8] Dimitris Chatzidimitriou, Archontia C. Giannopoulou, Spyridon Maniatis, Clément Requile, Dimitrios M. Thilikos, and Dimitris Zoros, *Fixed Parameter Algorithms for Completion Problems on Planar Graphs*, Algorithmic Graph Theory on the Adriatic Coast (AGTAC 2015), 2015.

[9] Dimitris Chatzidimitriou, Archontia C. Giannopoulou, Clément Requile, Dimitrios M. Thilikos and Dimitris Zoros, *A Fixed Parameter Algorithm for Plane Subgraph Completion*, 13th Cologne-Twente Workshop on Graphs & Combinatorial Optimization (CTW 2015), 2015.

Graph Theory

Textbooks

[10] Bela Bollobas, *Graph Theory, An Introductory Course*, Springer, 1979.

[11] Bela Bollobas, *Modern Graph Theory*, Springer, 1998.

[12] Adrian Bondy and U.S.R. Murty, *Graph Theory*, Springer, 2008.

[13] Reinhard Diestel, *Graph Theory*, GTM 173, 4th edition, Springer, 2010.

[14] Jonathan L. Gross, Jay Yellen, and Ping Zhang, *Handbook of Graph Theory*, CRC Press, 2013.

[15] Graph Theory, *Frank Harary*, Westview Press, 1994.

[16] T. Kloks, *Treewidth. computations and approximations*, Lecture Notes in Computer Science, 842, 1994.

[17] Dénes Kőnig, *Theorie der endlichen und unendlichen Graphen*, Leipzig: Akademische Verlagsgesellschaft, 1936.

Surveys

[18] S. Arnborg, *Efficient algorithms for combinatorial problems on graphs with bounded decomposability (a survey)*, BIT 25, pp. 2–33, 1985.

[19] Hans L. Bodlaender, *A tourist guide through treewidth*, Acta Cybernetica, 11, pp. l–23, 1993.

[20] Hans L. Bodlaender, *A partial k-arboretum of graphs with bounded treewidth (Tutorial)*, Theoretical Computer Science, Volume 209, Issues 1–2, pp. 1–45, 1998.

[21] Hans L. Bodlaender, *Treewidth: Structure and Algorithms*, International Colloquium on Structural Information and Communication Complexity (SIROCCO 2007), Structural Information and Communication Complexity, pp. 11-25, 2007.

[22] Josep Díaz, Jordi Petit, and Maria Serna, *A survey of graph layout problems*, ACM Computing Surveys (CSUR), Volume 34 Issue 3, pp. 313–356, 2002.

[23] B. A. Reed *Algorithmic Aspects of Tree Width* Recent Advances in Algorithms and Combinatorics Part of the series CMS Books in Mathematics, pp. 85–107, 2003.

Papers

[24] Noga Alon, Daniel Lokshtanov, and Saket Saurabh, *Fast Fast*, In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, CALP (1), Volume 5555 of Lecture Notes in Computer Science, Springer, pp. 49–58, 2009.

[25] K. Appel and W. Haken, *Every planar map is four colorable. Part I. Discharging*, Illinois J. Math., 21, pp. 429–490, 1977.

[26] K. Appel and W. Haken, *Every planar map is four colorable. Part II. Reducibility*, Illinois J. Math., 21, pp. 491–567, 1977.

[27] K. Appel and W. Haken, *The Solution of the Four-Color Map Problem* Sci. Amer. 237,pp. 108–121, 1977.

[28] K. Appel and W. Haken, *The Four Color Proof Suffices*, Math. Intell. 8, pp. 10–20 and 58, 1986

[29] K. Appel and W. Haken, *Every Planar Map is Four-Colorable*, Providence, RI: Amer. Math. Soc., 1989

[30] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski, *Complexity of Finding Embeddings in a k-Tree*, SIAM. J. on Algebraic and Discrete Methods, 8(2), pp. 277–284, 1987.

[31] Stefan Arnborg, Jens Lagergren, and Detlef Seese, *Easy problems for tree-decomposable graphs*, Journal of Algorithms, 12, pp. 308–340, 1991.

[32] Patrick Bellenbaum and Reinhard Diestel, *Two Short Proofs Concerning Tree-Decompositions*, Comb. Probab. Comput. 11, 6, pp. 541-547, 2002.

[33] Umberto Bertelé and Francesco Brioschi, *Nonserial Dynamic Programming*, Academic Press, 1972.

[34] Hans L. Bodlaender, *A linear time algorithm for finding tree-decompositions of small treewidth*, SIAM Journal on Computing, 25 (6), pp. 1305–1317, 1996.

[35] Hans L. Bodlaender, *Dynamic programming on graphs with bounded treewidth*, Automata, Languages and Programming: 15th International Colloquium Tampere, Proceedings, Springer Berlin Heidelberg, pp. 105–118, 1988.

[36] Hans L. Bodlaender, Jitender S. Deogun, Klaus Jansen, Ton Kloks, Dieter Kratsch, Haiko Müller, and Zsolt Tuza, *Rankings of graphs*, SIAM J. Discrete Math., 11(1),pp. 168–181 (electronic), 1998.

[37] Hans L. Bodlaender, Ton Kloks, *Efficient and Constructive Algorithms for the Pathwidth and Treewidth of Graphs*, Journal of Algorithms Volume 21, Issue 2, pp. 358–402, 1996.

[38] B. Bollobás, P. A. Catlin, and Paul Erdős, (1980), *Hadwiger's conjecture is true for almost every graph*, European Journal of Combinatorics, 1, pp. 195–199, 1980.

[39] Fan R. K. Chung, *On the cutwidth and the topological bandwidth of a tree*, SIAM Journal on Algebraic and Discrete Methods, 6(2), pp. 268–277, 1985.

[40] Moon Jung Chung, Fillia Makedon, Ivan Hal Sudborough, and Jonathan Turner, *Polynomial time algorithms for the MIN CUT problem on degree restricted trees*, SIAM Journal on Computing, 14(1),pp. 158–177, 1985.

[41] Dariusz Dereniowski, *From Pathwidth to Connected Pathwidth*, SIAM J. vol 26, 2012.

[42] M. DeVos, Z. Dvořák, J. Fox, J. McDonald, B. Mohar, and D. Scheide, *Minimum degree condition forcing complete graph immersion*, Combinatorica, Volume 34, Issue 3, pp. 279–298, 2014.

[43] Leonhard Euler, *Solutio problematis ad geometriam situs pertinentis*, Eneström 53, 1741.

[44] Fedor V. Fomin, Saket Saurabh, and Dimitrios M. Thilikos *Strengthening Erdös-Pósa property for minor-closed graph classes*, Journal of Graph Theory, Volume 66, Issue 3, pp. 235–240, 2011.

[45] Hugo Hadwiger, *Über eine Klassifikation der Streckenkomplexe*, Vierteljschr. Naturforsch. Ges. Zürich, 88, pp. 33–143, 1943.

[46] Marcin Kamiński,Daniël Paulusma, and Dimitrios M. Thilikos, *Contractions of Planar Graphs in Polynomial Time*, 18th Annual European Symposium on Algorithms ESA 2010, Lecture Notes in Computer Science, Springer, Vol. 6346, pp. 122–133, 2010.

[47] Marcin Kamiński and Dimitrios M. Thilikos, *Contraction checking in graphs on surfaces*, International Symposium on Theoretical Aspects of Computer Science, STACS 2012, 2012.

[48] T. Kashiwabara and T. Fujisawa, *NP-completeness of the problem of finding a minimum-clique-number interval graph containing*

a given graph as a subgraph, Proc. International Symposium on Circuits and Systems, pp. 657–660, 1979.

[49] Nancy G. Kinnersley, *The vertex separation number of a graph equals its path-width*, Information Processing Letters Volume 42, Issue 6, pp. 345–350, 1992.

[50] Ephraim Korach and Nir Solel, *Tree-width, path-width, and cutwidth*, Discrete Applied Mathematics, 43(1), pp. 97–101, 1993.

[51] Kazimierz Kuratowski, *Sur le problème des courbes gauches en topologie*, Fund. Math., pp. 271–283, 1930.

[52] Jens Lagergren, *The size of an interwine*, In Serge Abiteboul and Eli Shamir, editors, ICALP, Volume 820 of Lecture Notes in Computer Science, Springer, pp. 520–531, 1994.

[53] Thomas Lengauer, *Black-white pebbles and graph separation*, Acta Informatica, 16 (4), pp. 465–475, 1981.

[54] R. H. Möhring, *Graph problems related to gate matrix layout and PLA folding*, in Computational graph theory, vol. 7 of Comput. Suppl., Springer, Vienna, pp. 17–51, 1990.

[55] B. Monien and I. H. Sudborough, *Min cut is NP-complete for edge weighted trees*, Theoretical Computer Science, 58(1-3), pp. 209–229, 1988.

[56] T. Ohtsuki, H. Mori, E. Kuh, T. Kashiwabara, and T. Fujisawa, *One-dimensional logic gate assignment and interval graphs*, IEEE Transactions on Circuits and Systems, 26 (9), pp. 675–684, 1979.

[57] N. Robertson, D. P. Sanders, P. D. Seymour, and R. Thomas, *A New Proof of the Four Colour Theorem*, Electron. Res. Announc. Amer. Math. Soc. 2,pp. 17–25, 1996.

[58] Dimitrios M. Thilikos, Hans L. Bodlaender, and Michael R. Fellows, *Derivation of algorithms for cutwidth and related graph layout parameters*, Journal of Computer and System Sciences, 75(4), pp. 231–244, 2009.

[59] Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender, *Cutwidth I: A linear time fixed parameter algorithm*, Journal of Algorithms, 56(1), pp. 1–24, 2005.

[60] Robin Thomas, *A menger-like property of tree-width: The finite case* Journal of Combinatorial Theory, Series B, 48(1), pp. 67 – 76,1990.

[61] R. Thomas, *Tree-decompositions of graphs (lecture notes)*, School of Mathematics. Georgia Institute of Technology, Atlanta, Georgia 30332, USA, 1996.

[62] Mihalis Yannakakis, *A polynomial algorithm for the min-cut linear arrangement of trees* Journal of the ACM, Volume 32, Issue 4, pp. 950–988, 1985.

Graph Minors

Surveys

[63] Daniel Bienstock, *Algorithmic implications of the graph minor theorem*, Book Chapter, Handbooks in Operations Research and Management Science Volume 7, pp. 481–502, 1995.

[64] N. Robertson, P.D. Seymour, *Graph minors—a survey*, Surveys in combinatorics 1985, London Math. Soc. Lecture Note Ser., vol. 103,pp. 153–171,1985.

Graph Minors Series

[65] Neil Robertson, P.D. Seymour, *Graph minors. I. Excluding a forest*, Journal of Combinatorial Theory, Series B, Volume 35, Issue 1, pp. 39–61, 1983.

[66] Neil Robertson, P.D. Seymour, *Graph minors. II. Algorithmic aspects of tree-width*, Journal of Algorithms, Volume 7, Issue 3, pp. 309–322, 1986.

[67] Neil Robertson, P.D. Seymour, *Graph minors III: Planar tree-width*, Journal of Combinatorial Theory, Series B, Volume 36, Issue 1, pp. 49–64, 1984.

[68] Neil Robertson, P.D. Seymour, *Graph minors IV:Tree-width and well-quasi-ordering*, Journal of Combinatorial Theory, Series B Volume 48, Issue 2, pp. 227–254,1990.

[69] Neil Robertson, P.D. Seymour, *Graph minors. V. Excluding a planar graph*, Journal of Combinatorial Theory, Series B, Volume 41, Issue 1, pp. 92-114, 1986.

[70] Neil Robertson, P.D. Seymour, *Graph minors VI: Disjoint paths across a disc*, Journal of Combinatorial Theory, Series B, Volume 41, Issue 1, pp. 115–138,1986.

[71] Neil Robertson, P.D. Seymour, *Graph minors VII: Disjoint paths on a surface*, Journal of Combinatorial Theory, Series B, Volume 45, Issue 2, pp. 212–254, 1988.

[72] Neil Robertson, P.D. Seymour, *Graph minors VIII: A kuratowski theorem for general surfaces*, Journal of Combinatorial Theory, Series B, Volume 48, Issue 2, pp. 255–288, 1990.

[73] Neil Robertson, P.D. Seymour, *Graph minors IX: Disjoint crossed paths*, Journal of Combinatorial Theory, Series B, Volume 49, Issue 1, pp. 40–77, 1990.

[74] Neil Robertson, P.D. Seymour, *Graph minors X: Obstructions to tree-decomposition*, Journal of Combinatorial Theory, Series B, Volume 52, Issue 2, pp. 153–190, 1991.

[75] Neil Robertson, P.D. Seymour, *Graph minors XI: Circuits on a Surface tree-width*, Journal of Combinatorial Theory, Series B, Volume 60, Issue 1, pp. 72–106, 1994.

[76] Neil Robertson, P.D. Seymour, *Graph minors XII: Distance on a Surface*, Journal of Combinatorial Theory, Series B, Volume 64, Issue 2, pp. 240–272, 1995.

[77] Neil Robertson and P. D. Seymour *Graph minors. XIII. The disjoint paths problem*, Journal of Combinatorial Theory. Series B, Volume 63, Issue 1, pp. 65–110, 1995.

[78] Neil Robertson, P.D. Seymour, *Graph minors XIV: Extending an Embedding*, Journal of Combinatorial Theory, Series B, Volume 65, Issue 1, pp. 23–50, 1995.

[79] Neil Robertson, P.D. Seymour, *Graph minors XV: Giant Steps*, Journal of Combinatorial Theory, Series B, Volume 68, Issue 1, pp. 112–148, 1996.

[80] Neil Robertson and Paul D. Seymour, *Graph minors. XVI. Excluding a non-planar graph*, Journal of Combinatorial Theory. Series B, Volume 89, Isuue 1, pp. 43–76, 2003.

[81] Neil Robertson, P.D. Seymour, *Graph minors XVII: Taming a Vortex*, Journal of Combinatorial Theory, Series B, Volume 77, Issue 1, pp. 162–210,1999.

[82] Neil Robertson, P.D. Seymour, *Graph minors XVIII: Tree-decompositions and well-quasi-ordering*, Journal of Combinatorial Theory, Series B, Volume 89, Issue 1, pp. 77–108, 2003.

[83] Neil Robertson, P.D. Seymour, *Graph minors XIX: Well-quasi-ordering on a surface*, Journal of Combinatorial Theory, Series B, Volume 90, Issue 2, pp. 325–385, 2004.

[84] Neil Robertson and P. D. Seymour, *Graph minors. XX. Wagner's conjecture*, Journal of Combinatorial Theory. Series B, Volume 92, Issue 2,, pp. 325–357, 2004.

[85] Neil Robertson and Paul D. Seymour, *Graph minors. XXI. Graphs with unique linkages*, Journal of Combinatorial Theory. Series B, Volume 99, Issue 3, pp. 583–616, 2009.

[86] Neil Robertson and P. D. Seymour, *Graph minors. XXII. Irrelevant vertices in linkage problems*, Journal of Combinatorial Theory. Series B, Volume 102, Issue 2, pp. 530–563, 2012.

[87] Neil Robertson and Paul D. Seymour, *Graph minors XXIII. Nash-Williams' immersion conjecture*, Journal of Combinatorial Theory. Series B, Volume 100, Issue 2,pp. 181–205, 2010.

Papers

[88] Harvey Friedman, Neil Robertson and P. D. Seymour *The metamathematics of the graph minor theorem*, in Simpson, S., Logic and Combinatorics, Contemporary Mathematics, 65, American Mathematical Society, pp. 229–261, 1987.

[89] Ken-ichi Kawarabayashi and Paul Wollan, *A shorter proof of the graph minor algorithm: the unique linkage theorem*, In Leonard J. Schulman, editor, STOC, pp. 687–694, 2010.

[90] C.St.J.A. Nash-Williams, *On well-quasi-ordering infinite trees*, Proc. Comb. Philos. Soc. 61, pp. 697–720, 1965.

[91] C.St.J.A. Nash-Williams, *On well-quasi-ordering trees*, Theory of Graphs and Its Applications (Proc. Symp. Smolenice, 1963), Publ. House Czechoslovak Acad. Sci., pp. 83–84, 1964.

[92] Neil Robertson, Paul Seymour, and Robin Thomas, *Quickly excluding a planar graph*, Journal of Combinatorial Theory. Series B, Volume 62, Issue 2, pp. 323–348, 1994.

[93] K. Wagner, *Graphentheorie*, vol. 248/248a, B. J. Hochschultaschenbucher, Mannheim, p. 61, 1970.

[94] K. Wagner, *Über eine Eigenschaft der ebenen Komplexe*, Math. Ann., 114, pp. 570–590, 1937.

Graph Searching

Surveys

[95] B. Alspach, *Searching and sweeping graphs: a brief survey*, Matematiche (Catania), 59, pp. 5–37, 2006.

[96] D. Bienstock, *Graph searching, path-width, tree-width and related problems (a survey)*, Reliability of computer and communication networks, DIMACS Ser. Discrete Math. Theoret. Comput. Sci., vol. 5, 1989.

[97] Fedor V. Fomin, Dimitrios M. Thilikos, *An annotated bibliography on guaranteed graph searching*, Theoretical Computer Science 399, 2008.

[98] Stephan Kreutzer, *Graph Searching Games*, Book Chapter, Lectures in Game Theory for Computer Scientists, Edited by Krzysztof R. Apt and Erich Grädel, Cambridge University Press, 2011.

Papers

[99] Isolde Adler, Christophe Paul, and Dimitrios M. Thilikos, *Connected search against a lazy robber* , 8th workshop on GRAph Searching, Theory & Applications (GRASTA 2017), 2017.

[100] Lali Barrière, Paola Flocchini, Fedor V. Fomin, Pierre Fraigniaud, Nicolas Nisse, Nicola Santoro, Dimitrios M. Thilikos, *Connected graph searching*, Information and Computation 219, 2012.

[101] L. Barrière, P. Fraigniaud, N. Santoro, and D. M. Thilikos, *Searching is not jumping*, in Proceedings of the 29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2003), vol. 2880 of Lecture Notes in Comput. Sci., Springer, pp. 34–45, 2003.

[102] D. Bienstock and P. Seymour, *Monotonicity in graph searching*, J. Algorithms, 12, pp. 239-245, 1991.

[103] H. L. Bodlaender and D. M. Thilikos, *Computing small search numbers in linear time*, in Proceedings of the First International Workshop on Parameterized and Exact Computation (IWPEC 2004), vol. 3162 of Lecture Notes in Comput. Sci., Springer, pp. 37–48, 2004.

[104] R. Breisch, *An intuitive approach to speleotopology*, Southwestern Cavers (A publication of the Southwestern Region of the National Speleological Society), VI, pp. 72–78, 1967.

[105] N. D. Dendris, L. M. Kirousis, and D. M. Thilikos, *Fugitive-search games on graphs and related parameters*, Theoret. Comput. Sci., 172, pp. 233–254, 1997.

[106] F. V. Fomin and D. M. Thilikos, *On the monotonicity of games generated by symmetric submodular functions*, Discrete Appl. Math., 131, pp. 323–335, 2003.

[107] P. Fraigniaud and N. Nisse, *Monotony properties of connected visible graph searching*, in Proceedings of the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2006), vol. 4271 of Lecture Notes in Comput. Sci., Springer, pp. 229–240, 2006.

[108] P. A. Golovach, *A topological invariant in pursuit problems (Russian)*, Differentsial′nye Uravneniya, 25, 923–929. 1097, 1989. translation in Differential Equations 25, no. 6, pp. 657–661, 1989.

[109] P. A. Golovach, *Equivalence of two formalizations of a search problem on a graph (Russian)*, Vestnik Leningrad. Univ. Mat. Mekh. Astronom., pp. 10–14, 1989. translation in Vestnik Leningrad Univ. Math. 22, no. 1,pp. 13–19, 1989.

[110] R. Isaacs, *Differential games*, A mathematical theory with applications to warfare and pursuit, control and optimization, John Wiley & Sons Inc., New York, 1965.

[111] L. M. Kirousis and C. H. Papadimitriou, *Interval graphs and searching*, Discrete Math., 55, pp. 181–184, 1985.

[112] L. M. Kirousis and C. H. Papadimitriou, *Searching and pebbling*, Theoret. Comput. Sci., 47, pp. 205–218, 1986.

[113] A. S. LaPaugh, *Recontamination does not help to search a graph*, J. Assoc. Comput. Mach., 40, pp. 224–245, 1993.

[114] T. Lengauer, *Black-white pebbles and graph separation*, Acta Informatica, Volume 16, Issue 4, pp. 465–475, 1981.

[115] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou, *The complexity of searching a graph*, J. Assoc. Comput. Mach., 35, pp. 18–44, 1988.

[116] T. D. Parsons, *Pursuit-evasion in a graph*, in Theory and applications of graphs, vol. 642 of Lecture Notes in Math., Springer, Berlin, pp. 426–441, 1978.

[117] N. N. Petrov, *A problem of pursuit in the absence of information on the pursued*, Differentsial′nye Uravneniya, 18, pp. 1345–1352, 1982.

[118] N. N. Petrov, *The Cossack-robber differential game*, Differentsial′nye Uravneniya, 19, pp. 1366–1374, 1983.

[119] A. Takahashi, S. Ueno, and Y. Kajitani, *Minimal forbidden minors for the family of graphs with proper-path-width at most two*, IEICE Trans. Fund. E78-A, pp. 1828–1839, 1995.

[120] D. M. Thilikos, *Algorithms and obstructions for linear-width and related search parameters*, Discrete Appl. Math., 105, pp. 239–271, 2000.

[121] P. D. Seymour and R. Thomas, *Graph searching and a min-max theorem for tree-width*, J. Combin. Theory Ser. B, 58, pp. 22–33, 1993.

[122] B. Yang, D. Dyer, and B. Alspach, *Sweeping graphs with large clique number*, in Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC 2004), vol. 3341 of Lecture Notes in Comput. Sci., Springer, pp. 908–920, 2004.

Obstruction Sets

Papers

[123] Isolde Adler, Martin Grohe, and Stephan Kreutzer, *Computing excluded minors*, In Shang-Hua Teng, editor, SODA, pp. 641–650, 2008.

[124] S. Arnborg, A. Proskurowski, and D. G. Corneil, *Forbidden minors characterization of partial 3-trees*, Discrete Math., Volume 80, Issue 1, pp. 1–19, 1990.

[125] Kevin Cattell, Michael J. Dinneen, Rodney G. Downey, Michael R. Fellows, and Michael A. Langston, *On computing graph minor obstruction sets*, Theor. Comput. Sci., 233(1-2), pp. 107–127, 2000.

[126] F. R. K. Chung and Paul D. Seymour, *Graphs with small bandwidth and cutwidth*, Discrete Mathematics, 75(1-3):pp. 113–119, 1989.

[127] Bruno Courcelle, Rodney G. Downey, and Michael R. Fellows, *A note on the computability of graph minor obstruction sets for monadic second order ideals*, J. UCS, 3(11), pp. 1194–1198, 1997.

[128] Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi, *Algorithmic Graph Minor Theory: Improved Grid Minor Bounds and Wagner's Contraction*, International Symposium on Algorithms and Computation (ISAAC 2006), pp. 3–15, 2006.

[129] Zdenek Dvorak, Archontia C. Giannopoulou and Dimitrios M. Thilikos, *Forbidden graphs for tree-depth*, European J. Combin. 33 (5), pp. 969–979, 2012.

[130] Archontia C. Giannopoulou, Michal Pilipczuk, Jean-Florent Raymond, Dimitrios M. Thilikos, and Marcin Wrochna, *Cutwidth: Obstructions and Algorithmic Aspects*, 11th International Symposium on Parameterized and Exact Computation (IPEC 2016), pp. 1–13, 2017.

[131] Archontia C. Giannopoulou, Michal Pilipczuk, Jean-Florent Raymond, Dimitrios M. Thilikos, and Marcin Wrochna, *Cutwidth: Obstructions and Algorithmic Aspects*, CoRR abs/1606.05975, 2016.

[132] Marcin Kaminski, Daniël Paulusma, and Dimitrios M. Thilikos, *Contracting planar graphs to contractions of triangulations*, Journal of Discrete Algorithms, Volume 9, Issue 3, pp. 299–306, 2011.

[133] Nancy G. Kinnersley, Michael A. Langston, *Obstruction set isolation for the gate matrix layout problem*, Discrete Applied Mathematics, Volume 54, Issues 2-3, pp. 169–213, 1994.

[134] A. Satyanarayana and L. Tung, *A characterization of partial 3-trees*, Networks 20(3), pp. 299–322, 1990.

[135] A. Takahashi, S. Ueno, and Y. Kajitani, *Minimal acyclic forbidden minors for the family of graphs with bounded path-width*, Discrete Mathematics, Volume 127, Issues 1-3, pp. 293–304, 1994.

[136] A. Takahashi, S. Ueno, and Y. Kajitani, *Minimal forbidden minors for the family of graphs with proper-path-width at most two*, IEICE Trans. Fund. E78-A, pp. 1828–1839, 1995.

[137] Paul Wollan, *The structure of graphs not admitting a fixed immersion*, CoRR, abs/1302.3867, 2013.

Parameterized Complexity

Textbooks

[138] Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *The Multivariate Algorithmic Revolution and Beyond*, Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday, Volume 7370 of Lecture Notes in Computer Science, Springer, 2012.

[139] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh, *Parameterized Algorithms*, Springer, 2015.

[140] Rodney G. Downey and Michael R. Fellows, *Fundamentals of Parameterized Complexity*, Texts in Computer Science, Springer, 2013.

[141] Rodney G. Downey and Michael R. Fellows, *Parameterized Complexity*, Springer, 1998.

[142] Jörg Flum and Martin Grohe, *Parameterized Complexity Theory*, Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

[143] Rolf Niedermeier, *Invitation to fixed-parameter algorithms*, Volume 31 of Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, Oxford, 2006.

Surveys

[144] H. L. Bodlaender, P. Heggernes, and D. Lokshtanov, *Graph modification problems*, Dagstuhl seminar 14071, Dagstuhl Reports, Volume 4, Issue 2, pp. 38–59, 2014.

[145] Jiong Guo and Rolf Niedermeier, *Invitation to data reduction and problem kernelization* SIGACT News, Volume 38, Issue 1, pp. 31–45, 2007.

[146] Christian Komusiewicz, André Nichterlein, and Rolf Niedermeier, *Parameterized Algorithmics for Graph Modification Problems: On Interactions with Heuristics*, International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2015), pp. 3–15, 2015.

[147] Stefan Kratsch, *Recent developments in kernelization: A survey*, Bulletin of the EATCS, 113, 2014.

[148] Yunlong Liu, Jianxin Wang, and Jiong Guo *An Overview of Kernelization Algorithms for Graph Modification Problems*, Tsinghua Science and Technology, Volume 19, Issue 4, 2014.

Papers

[149] Karl R. Abrahamson and Michael R. Fellows, *Finite automata, bounded treewidth and well-quasiordering* AMS Summer

Workshop on Graph Minors, Graph Structure Theory, Contemporary Mathematics vol. 147, pp. 539–564, 1993.

[150] Faisal N. Abu-Khzam and Henning Fernau, *Kernels: Annotated, proper and induced*, In Proc. 2nd IWPEC, Volume 4169 of LNCS, pp. 264–275, 2006.

[151] Rémy Belmonte, Pim van 't Hof, Marcin Kaminski, Daniël Paulusma, and Dimitrios M. Thilikos, *Characterizing graphs of small carving-width*, Discrete Applied Mathematics, 161(13-14), pp. 1888–1893, 2013.

[152] Hans L. Bodlaender and Babette van Antwerpen-de Fluiter *Reduction algorithms for graphs of small treewidth*, Inf. Comput., 167, pp. 86–119, 2001.

[153] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin, *On problems without polynomial kernels*, J. Comput. Syst. Sci., 75(8), pp. 423–434, 2009.

[154] Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos, *(Meta) Kernelization*, In Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2009), pp. 629–638, 2009.

[155] Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos, *(Meta) Kernelization*, J. ACM 63(5), pp. 1–69, 2016.

[156] Richard B. Borie, R. Gary Parker, and Craig A. Tovey, *Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families*, Algorithmica, 7, pp. 555–581, 1992.

[157] Jonathan F. Buss and Judy Goldsmith, *Nondeterminism within* P, SIAM J. Comput., 22(3), pp. 560–572, 1993.

[158] Liming Cai and Jianer Chen, *On fixed-parameter tractability and approximability of NP optimization problems*, Journal of Computer and System Sciences, Volume 54, pp. 465–474, 1997.

[159] Marco Cesati and Luca Trevisa, *On the efficiency of polynomial time approximation schemes*, Information Processing Letters Volume 64, Issue 4, pp. 165–171, 1997.

[160] J. Chen, X. Huang, I.A. Kanj, and G. Xia *Polynomial time approximation schemes and parameterized complexity*, Discrete Applied Mathematics Volume 155, Issue 2, pp. 180-193. 2007.

[161] J. Chen, I.A. Kanj, and G. Xia, *Improved upper bounds for vertex cover*, Theoretical Computer Science, Volume 411, Issues 40–42, pp. 3736–3756, 2010.

[162] J. Chen, I.A. Kanj, and G. Xia, *Simplicity is beauty: Improved upper bounds for vertex cover*, Technical Report TR05-008, School of CTI, DePaul University, 2005.

[163] Rajesh Hemant Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk, *Designing FPT algorithms for cut problems using randomized contractions*, In 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012), pp. 460–469, 2012.

[164] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M, van Rooij, and Jakub Onufry Wojtaszczyk, *Solving connectivity problems parameterized by treewidth in single exponential time*, In Rafail Ostrovsky, editor, FOCS, pp. 150–159, 2011.

[165] Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos, *Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs*, Journal of the ACM, 52(6), pp. 866–893, 2005.

[166] Erik D. Demaine and MohammadTaghi Hajiaghayi, *Bidimensionality: new connections between FPT algorithms and PTASs*, Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms (SODA 05), pp. 590–601, 2005.

[167] Rodney G. Downey and Michael R. Fellows, *Fixed-parameter tractability and completeness I: Basic results*, SIAM J. Comput., Volume 24, series 4, pp. 873–921, 1995.

[168] Rodney G. Downey and Michael R. Fellows, *Fixed-parameter tractability and completeness II: On completeness for W[1]*, Theor. Comput. Sci., Volume 141, series1&2, pp. 109–131, 1995.

[169] Michael R. Fellows and Bart M. P. Jansen, *FPT Is Characterized by Useful Obstruction Sets*, International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2013), pp. 261–273, 2013.

[170] Henning Fernau, Till Fluschnik, Danny Hermelin, Andreas Krebs, Hendrik Molter, and Rolf Niedermeier, *Diminishable parameterized problems and strict polynomial kernelization* CoRR, abs/1611.03739, 2016.

[171] Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh, *Planar F-Deletion: Approximation, Kernelization and Optimal FPT Algorithms*, In Proceedings of the 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science (FOCS '12), pp. 470-479, 2012.

[172] Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh, *Planar F-Deletion: Approximation, Kernelization and Optimal FPT Algorithms*, CoRR abs/1204.4230, 2012.

[173] Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh, *Bidimensionality and EPTAS*, In 22st ACM–SIAM Symposium on Discrete Algorithms (SODA 2011), pp. 748–759, 2011.

[174] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos, *Bidimensionality and Kernels*, SODA 2010, pp. 503–510, 2010.

[175] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos, *Bidimensionality and Kernels*, CoRR abs/1606.05689, 2016.

[176] Valentin Garnero, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. *Explicit linear kernels via dynamic programming*, SIAM J. Discrete Math., 29(4), pp. 1864–1894, 2015.

[177] Archontia C. Giannopoulou, Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh, *Uniform kernelization complexity of hitting forbidden minors*, CoRR, abs/1502.03965, 2015.

[178] Archontia C. Giannopoulou, Michal Pilipczuk, Dimitrios M. Thilikos, Jean-Florent Raymond, and Marcin Wrochna, *Linear kernels for edge deletion problems to immersion-closed graph classes*, (To appear in ICALP 2017), CoRR, abs/1609.07780, 2016.

[179] Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan, *Finding topological subgraphs is fixed-parameter tractable*, In STOC, pp. 479–488, 2011.

[180] Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar, *Linear Kernels and Single-Exponential Algorithms Via Protrusion Decompositions*, ACM Trans. Algorithms 12(2), pp. 1-41, 2016.

[181] Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh, *Lossy kernelization*, In Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, (STOC 2017), pp. 224–237, 2017.

[182] Dimitrios M. Thilikos, *Bidimensionality and parameterized algorithms (invited talk)*, In 10th International Symposium on Parameterized and Exact Computation (IPEC 2015), pp. 1–16, 2015.

[183] Siamak Tazari, *Faster approximation schemes and parameterized algorithms on H-minor-free and odd-minor-free graphs*, Mathematical Foundations of Computer Science 2010: 35th International Symposium (MFCS 2010), pp. 641–652, 2010.

Logic and Complexity

Textbooks

[184] B. Courcell and Joost Engelfriet, *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach (1st ed.)*, Cambridge University Press, 2012.

[185] Sanjoy Dasgupta, Christos Papadimitriou, and Umesh Vazirani, *Algorithms*, McGraw-Hill Education, 2006.

[186] Herbert B. Enderton, *A mathematical introduction to logic*, Academic Press, 1972.

[187] Michael R. Garey and David S. Johnson, *Computers and intractability. A guide to the theory of NP-completeness*, W. H. Freeman and Co., San Francisco, Calif., 1979.

[188] Elliott Mendelson, *Introduction to mathematical logic (3. ed.)*. Chapman and Hall, 1987.

Papers

[189] B. Courcelle, *Graph rewriting: an algebraic and logic approach*, Handbook of theoretical computer science (vol. B), pp. 193 - 242, 1990.

[190] B. Courcelle, *The expression of graph properties and graph transformations in monadic second-order logic*, Handbook of Graph Grammars, pp. 313–400, 1997.

[191] B. Courcell, *The monadic second-order logic of graphs I: Recognizable sets of finite graphs*, Information and Computation, 85,pp. 12–75, 1990.

[192] Michael R. Fellows and Michael A. Langston, *Nonconstructive tools for proving polynomial-time decidability*, J. Assoc. Comput. Mach., Volume 35, Issue 3, pp. 727–739, 1988.

[193] Michael R. Fellows and Michael A. Langston, *On search, decision, and the efficiency of polynomial-time algorithms*, J. Comput. System Sci., Volume 49, Issue 3, pp. 769–779, 1994.

Other

[194] J. Hajek, *K:)nigsberg bridges solved*, http://www.matheory.info/konigsberg/

[195] Νικος Kazantzakης, *Ταξιδεύοντας: Ρουσία*, Εκδόσεις Καζαντζάκη, 1956.

[196] Nikos Kazantzakis, *Russia: A Chronicle of Three Journeys in the Aftermath of the Revolution*, Creative Arts Book Company, 1956.

[197] Milan Kundera, *The Festival of Insignificance*, Harper, 2013.

[198] James Joseph Sylvester, *Chemistry and Algebra*, Nature, 17, 284, 1878.

[199] Wolfram MathWorld, *Seven Bridges of Königsberg Problem*, http://mathworld.wolfram.com/KoenigsbergBridgeProblem.html

LIST OF SYMBOLS

Symbol	Explanation	Page
$(\mathbf{p}, r)$-$\mathsf{dist}(G)$	The function measuring the number of vertices we have to delete from G in order for $\mathbf{p}$ to have value at most r	11, 68
$\mathbb{N}$	The set of natural numbers	21
$\mathbb{N}^+$	The set of positive natural numbers	21
$\mathbb{Z}$	The set of integers	21
$\mathbb{Z}^+$	The set of positive integers	21
$\mathbb{R}$	The set of real numbers	21
$\mathbb{R}^+$	The set of positive real numbers	21
$[n]$	The set $\{1, 2, \ldots, n\}$	21
2^S	The power-set of S	21
$\binom{S}{2}$	The set of all subsets of S with cardinality 2	21
$f\|_S$	$f\|_S = \{(x, f(x)) \mid x \in S \cap A\}$	22
$f \setminus S$	$f \setminus S = \{(x, f(x)) \mid x \in A \setminus S\}$	22
$\varnothing$	The empty function	22

Symbol	**Explanation**	**Page**
$V(G)$	The vertex set of G	22
$E(G)$	The edge set of G	22
$n(G)$	The number of vertices of G	22
$\|G\|$	The number of vertices of G	22
$m(G)$	The number of edges of G	22
$G[S]$	The subgraph of G induced by $S \subseteq V(G)$	22
$G[F]$	The subgraph of G induced by $F \subseteq E(G)$	22
$G \setminus S$	The graph $G[V(G) \setminus S]$	22
$N_G(u)$	The neighbourhood of vertex u in G	22
$\deg_G(u)$	The degree of vertex u in G	22
$N_G(S)$	The neighbourhood of $S \subseteq V(G)$	22
$N_G[S]$	The closed neighbourhood of $S \subseteq V(G)$	22
$\partial_G(S)$	The set of vertices of S that are incident to edges not in $G[S]$	22
$\mathsf{Leaf}(T)$	The set of leaves of T	24
aTb	The unique path in T with end points a and b	24
K_k	Clique of k vertices	24
$K_{k,l}$	The graph $(A \cup B, \big\{\{u,v\} \mid u \in A \text{ and } v \in B\big\})$ for disjoint A, B	24
$L(G)$	The line graph of G	24
$G_1 \cup G_2$	The union of G_1 and G_2	24
$G_1 + G_2$	The disjoint union of G_1 and G_2	24
$G_1 \times G_2$	The lexicographic product of G_1 and G_2	24
$\boxplus_k$	The $(k \times k)$-grid	24

Symbol	Explanation	Page
$G \setminus u$	The graph obtained after the deletion of vertex u in G	26
$G \setminus e$	The graph obtained after the deletion of edge e in G	26
G/e	The graph obtained after the contraction of edge e in G	26
G/u	The graph obtained after the dissolution of vertex u in G	26
$\leq$	The subgraph relation	27
$\leq_{\text{sp}}$	The spanning subgraph relation	27
$\leq_{\text{in}}$	The induced subgraph relation	28
$\leq_{\text{c}}$	The contraction relation	28, 219
$\leq_{\text{tp}}$	The topological-minor relation	28
$\leq_{\text{m}}$	The minor relation	28, 138
$\leq_{\text{im}}$	The immersion relation	28
$\mathcal{G}$	The class of all graphs	29
$\mathsf{obs}_{\preceq}(\mathcal{C})$	The obstruction set of $\mathcal{C}$ for the relation $\preceq$	30
$G(\Gamma)$	The abstract graph defined from Γ	32
$F(\Gamma)$	The set of faces of Γ	32
$\text{ar}(R_i)$	The arity of the predicate R_i	38
$L^{\mathcal{G}}$	The language of graphs	39
$\mathfrak{A} \models \phi$	$\mathfrak{A}$ is a model of ϕ	39
$\mathsf{card}_{q,r}(S)$	The atomic formula testing whether the cardinality of S is equal to q modulo r	41
Σ^*	The set of strings over Σ, the Kleene closure of Σ	44
$\lvert w \rvert$	The size of the word w	45

Symbol	Explanation	Page
P	The class of polynomially solved problems	45
NP	The class of polynomially verified problems	45
p-Π	The parameterized version of Π	47
FPT	The class of FPT-problems	49
p-$\Pi \Subset \mathcal{C}$	The restriction of problem p-Π to graph class $\mathcal{C}$	51
$\mathsf{dom}(\mathbf{p})$	The dominion of $\mathbf{p}$	57
$\mathcal{G}[\mathbf{p}, k]$	The class containing the graphs with $\mathbf{p}(G) \leq k$	57
$\mathsf{obs}_{\preceq}(\mathbf{p}, k)$	The obstruction set of $\mathcal{G}[\mathbf{p}, k]$	57
$\mathcal{L}_{\mathrm{V}}(G)$	The set of all vertex layouts of G	60
$S^t_\sigma(i)$	The tree-supporting set of σ in position i	60
$\mathsf{cost}_t(G, \sigma)$	The "tree" cost of a vertex layout of G	60
$\mathbf{tvs}(G)$	The tree vertex separation number of G	60
$\mathsf{width}(T, B)$	The width of a tree-decomposition	61
$\mathbf{tw}(G)$	The treewidth of G	61
$S^p_\sigma(i)$	The path-supporting set of σ in position i	63
$\mathsf{cost}_p(G, \sigma)$	The "path" cost of a vertex layout of G	63
$\mathbf{pvs}(G)$	The vertex separation number of G	63
$\mathbf{pw}(G)$	The pathwidth of G	63
$\partial G(\sigma, i)$	The cut at position i	64
$\mathsf{cost}_c(G, \sigma)$	The "cut" cost of a vertex layout of G	64
$\mathbf{cw}(G)$	The cutwidth of G	64
$\mathcal{L}_{\mathrm{E}}(G)$	The set of all edge layouts of G	65
$S^l_\sigma(i)$	The line-supporting set of σ in position i	66
$\mathsf{cost}_l(G, \sigma)$	The "line" cost of an edge layout of G	66
$\mathbf{lw}(G)$	The linearwidth of G	66

Symbol	Explanation	Page
$\mathbf{vc}(G)$	The size of the minimum vertex cover of G	67
$\mathbf{fv}(G)$	The size of the minimum feedback vertex set of G	67
$\mathbf{pl}(G)$	The minimum number of edges we have to delete from G in order to make it planar	67
$\mathcal{G}_{\mathcal{F},k}$	The class containing all graphs G for which there exists a subset $S \subseteq V(G)$, of size at most k, such that $G \setminus S$ contains no graph from $\mathcal{F}$ as a minor	70
$H(d)$	The set of hyperplanes of $\mathbb{R}^d$	74
$S(d)$	The set of hyperspheres of $\mathbb{R}^d$	75
$\mathcal{E}_d(G)$	An embedding of G in the euclidean space $\mathbb{R}^d$	75
$\mathbf{E}_d(G)$	Then set of essential Embedding of G in $\mathbb{R}^d$	75
$\partial_G(\mathcal{E}_d(G), \Pi)$	The set of curves of $\mathcal{E}_d(G)$ that are intersected by Π	76
$\partial_G(\mathcal{E}_d(G), \Sigma)$	The set of curves of $\mathcal{E}_d(G)$ that are intersected by Σ	76
$\mathbf{cw}_d(G)$	The d-cutwidth of G	76
$\mathcal{E}_1(G)$	An embedding of G in $\mathbb{R}$	77
$\mathbf{scw}_d(G)$	The spherical d-cutwidth of G	80
$\mathcal{F}_1 \leq \mathcal{F}_2$	Ordering between two finite sets $\mathcal{F}_1, \mathcal{F}_2$ of graphs	95
$\mathcal{T}_k$	The class of graphs of treewidth at most k	97
$\mathcal{C}_{\mathcal{T}_k}$	the class of tree-dec expansions of a graph $G \in \mathcal{T}_k$	97
$\mathsf{width}(\mathcal{C})$	The width of an immersion-closed graph class $\mathcal{C}$	98
$\mathsf{dec}(\mathbf{p})$	The protrusion decomposability constant	112
$\mathcal{B}^{(t)}$	The set of all t-boundaried graphs	113

Symbol	Explanation	Page
$\mathcal{B}^{(\leq t)}$	The set $\bigcup_{t' \in \{0\} \cup [t]} \mathcal{B}^{(t')}$	113
$\mathcal{T}^{(\leq t)}$	The set of all boundaried graphs in $\mathcal{B}^{(\leq t)}$ whose underlying graph has treewidth at most $t-1$	113
$\psi_{\mathbf{G}}$	The label normalizing function of $\mathbf{G}$	113
$\oplus$	The "gluing" operation on boundaried graphs	114
$\equiv_{\mathbf{p},t}$	The equation relation on boundaried graphs	114
$\mathsf{null}(\mathbf{p})$	The equivalence class of all graphs G, such that $G \notin \mathsf{dom}(\mathbf{p})$	114
$\mathsf{rep}_{\mathbf{p}}(\mathbf{G})$	The (unique) boundaried graph in a representative collection $\mathcal{R}$ equivalent to $\mathbf{G}$	114
$\mathsf{card}_{\mathbf{p}}$	The function giving the number of equivalence classes of $\equiv_{\mathbf{p},t}$	115
$\mathbf{p}_{p\text{-}\Pi}$	$\mathbf{p}_{p\text{-}\Pi}(G) = \bullet\{k \mid (G,k) \in p\text{-}\Pi\}$ where $\bullet$ is min or max depending on the type of p-Π	116
$\mathsf{sol}_{p\text{-}\Pi}(G)$	$\mathsf{sol}_{p\text{-}\Pi}(G) = S$ where $\lvert S\rvert = \mathbf{p}_{p\text{-}\Pi}(G)$ and $(G,S) \in p$-Π, if such S exists	117
$\mathsf{Leaf}(T,r)$	The set of leaves of (T,r) other than r	132
$a \leq_{T,r} b$	b is a descendant of a in (T,r)	132
$a \neq_{T,r} b$	a and b are uncomperable in (T,r)	132
$\mathsf{desc}_{T,r}(q)$	The set of descendants of q in (T,r)	132
$\mathsf{child}_{T,r}(q)$	The children of q in (T,r)	132
$\mathsf{depth}_{T,r}(v)$	The depth of v in (T,r)	133
$\mathsf{height}_{T,r}(v)$	The height of v in (T,r)	133
$\mathsf{inner}_{T,r}(a,b)$	The inner part of (a,b)	133
$\mathsf{outer}_{T,r}(a,b)$	The outer part of (a,b)	133
$\mathsf{capacity}_{T,r}(a,b)$	The capacity of (a,b)	133

Symbol	Explanation	Page
$(T, r) \setminus (a, b)$	The tree obtained after an (a, b)-compression	134
$\mathsf{transp}_{\mathbf{p}}$	The transposition function of **p**	136
$\mathbf{G} \setminus S$	The boundaried graph obtained from **G** after the deletion of the vertices of S	138
$\mathbf{tw}(\mathbf{G})$	The treewidth of **G**	140
$\mathsf{gohigher}_{T,r}(v, \mu)$	The ancestor u of v such that $\min\{\mu, \mathsf{depth}_{T,r}(v) - \mathsf{depth}_{T,r}(u)\}$ is maximized	164
$\mathcal{A}(G, Y)$	The set containing the vertex set of the augmented connected components for (G, Y)	167
$\mathsf{potential}_{(\mathbf{G},D)}(a, b)$	The potential of a transition pair (a, b) of $(\mathbf{G}, D)$	172
$\mathsf{potential}_{(\mathbf{G},D)}(\mathcal{P})$	The potential of the transition collection $\mathcal{P}$ of $(\mathbf{G}, D)$	173
$\mathsf{p}(v)$	Placing a searcher on v	194
$\mathsf{r}(v)$	Removing a searcher from v	194
$\mathsf{s}(v, u)$	Sliding a searcher along $\{v, u\}$	194
$E(\mathcal{S}, i)$	The set of clean edges after i steps of $\mathcal{S}$	194
$\mathsf{cost}_G(\mathcal{S})$	The cost of search strategy Σ	195
$\mathbf{es}(G)$	The edge search number of G	195
$\mathbf{ns}(G)$	The node search number of G	196
$\mathbf{ms}(G)$	The mixed search number of G	196
$\mathbf{mes}(G)$	The monotone edge search number of G	197
$\mathbf{mns}(G)$	The monotone node search number of G	198
$\mathbf{mms}(G)$	The monotone mixed search number of G	198
$\mathbf{ces}(G)$	The connected edge search number of G	199
$\mathbf{cns}(G)$	The connected node search number of G	199

Symbol	Explanation	Page
$\mathbf{cms}(G)$	The connected mixed search number of G	199
$\mathbf{cmes}(G)$	The connected and monotone edge search number of G	200
$\mathbf{cmns}(G)$	The connected and monotone node search number of G	200
$\mathbf{cmms}(G)$	The connected and monotone mixed search number of G	200
$\mathbf{ilns}(G)$	The node search number of G, where the fugitive is invisible but lazy	202
$\mathsf{prefsec}(\mathcal{A})$	The ordering of prefixes of $\mathcal{A}$	206
$\circ$	The concatenation of orderings	206
$\mathbf{rev}(\mathbf{G})$	The reversed rooted graph $(G, S^{\text{out}}, S^{\text{in}})$	207
$\mathbf{enh}(G, S^{\text{in}}, S^{\text{out}})$	The enhancement of $(G, S^{\text{in}}, S^{\text{out}})$	207
$\partial_G(F)$	The boundary of the edge set F	209
$\mathsf{cost}_G(\mathcal{E}, i)$	The cost of an expansion $\mathcal{E}$ at position i on G	210
$\mathsf{cost}_G(\mathcal{E})$	The cost of an expansion $\mathcal{E}$ on G	210
$\mathbf{p}(G, S^{\text{in}}, S^{\text{out}})$	Minimum cost over all expansions	210
$\mathbf{mp}(G, S^{\text{in}}, S^{\text{out}})$	Minimum cost over all monotone expansions	210
$\mathbf{cmp}(G, S^{\text{in}}, S^{\text{out}})$	Minimum cost over all connected and monotone expansions	210
$\mathbf{cmp}(G)$	$\mathbf{cmp}(G, \emptyset, \emptyset)$	210
$\mathbf{glue}(\mathbf{G}_1, \ldots, \mathbf{G}_r)$	The "gluing" operation on r rooted graphs	212

GLOSSARY OF TERMS

Symbols