Tournaments and Optimality:
New Results in Parameterized Complexity

Michał Pilipczuk

Dissertation for the degree of Philosophiae Doctor (PhD)

University of Bergen, Norway

August 2013

Contents

Part I

Introduction

Chapter 1

A gentle introduction to parameterized complexity

1.1 The phenomenon of NP-completeness

Since Cook's seminal proof of NP-hardness of the SAT problem [75], the pursuit of understanding the border between efficiently tractable problems (i.e., languages in P) and problems that seem to require exponential-time computations (i.e., NP-hard languages) has driven most of the research in theoretical computer science. Despite the fact that a formal proof that P≠NP is still out of our reach, arguably we have a fairly good intuitive understanding of what features make a particular problem NP-hard. Starting with Karp's list of 21 problems [203], the net of known NP-complete problems has quickly reached volumes of books, see for reference the monograph of Garey and Johnson [161]. Today's research on NP-hardness focuses mostly on (i) understanding the source of intractability of a given problem, and (ii) identifying the most efficient ways of circumventing it by examining the problem's structure.

Arguably the most natural way of coping with an NP-hard problem is via **approximation**. In this framework, given an NP-hard optimization problem we relax the request of finding an optimal solution to finding a solution that is provably not far from being optimal; for instance, we may ask for a solution whose cost is at most twice as large as the optimal one. Generally, for a minimization problem an algorithm returning a value at most c times larger than the optimal one is called a *c-approximation*; this definition can be easily translated to maximization problems as well. Relaxation of the request of finding an optimal solution enables us to perform faster computations; the classical definition of an approximation algorithm assumes that it works in polynomial time.

It appears that the landscape of NP-complete problems that is uniformly flat when simple NP-hardness reductions are considered, turns out to be rich and varied when one examines the possibilities of designing approximation algorithms. Some problems, like Clique, cannot be approximated within multiplicative factor of $n^{1-\varepsilon}$ for any $\varepsilon > 0$, unless P=NP [310]. Other, like Dominating Set, admit an $\mathcal{O}(\log n)$ approximation that is optimal, again under P≠NP [12]. For a large number of problems, like Vertex Cover, a constant factor approximation is a limit [114]. Finally, some rare problems, like Knapsack, admit *polynomial-time approximation schemes* (PTASes), i.e. for every $\varepsilon > 0$ there exists a polynomial-time $(1+\varepsilon)$-approximation, whose running time can of course depend on ε. The systematic study of approximation algorithms exhibited both a number of useful and practical algorithmic techniques, mostly connected to ubiquitous usage of linear programming,

and deep connections with probability theory, convex geometry and discrete Fourier analysis, which are foundations of the theory of approximation lower bounds. We refer to the books of Vazirani [306] and of Williamson and Shmoys [309] for a deeper introduction to approximation algorithms.

A different approach is to accept the necessity of the exponential blow-up in the complexity that follows from NP-hardness, but try to design algorithms that keep this blow-up as small as possible. Here, we arrive at the field of **exact algorithms**. Consider for example the classical Clique problem: while the trivial algorithm that tries all possible subsets of vertices runs in $\mathcal{O}^*(2^n)$[1] time, one can reduce this running time to as low as $\mathcal{O}(1.211^n)$ via clever branching strategies [291]. Usually, the research in exponential-time algorithms focuses on lowering the base of the exponent as much as possible; however, for many problems even breaking the trivial 2^n barrier of brute-force enumeration or dynamic programming on subsets is a challenging task [33, 35, 37, 93, 95]. It also appears that algebraic techniques, in particular the inclusion-exclusion principle, subset convolutions and polynomial identity testing using the Schwartz-Zippel lemma, are very useful in the context of exact algorithms [33, 34, 36]. These algebraic tools can be also used to reduce the space complexity of exponential-time algorithms to polynomial [244, 257]. We refer to the book of Fomin and Kratsch [136] for more information on the field of exact algorithms.

1.2 Parameterized complexity in a nutshell

In this thesis we will be mostly interested in another paradigm, namely **parameterized complexity**. Informally speaking, the parameterized approach aims at identifying the source of intractability of a problem by introducing an auxiliary measure that reflects hardness of a given instance. The idea is to pinpoint the properties of the problem that make it hard to tackle, and then to use this understanding to design algorithms that perform efficiently assuming that the given instance has a 'weak spot', i.e., one of the hardness measures is small. Formally, in the parameterized setting an instance of a problem comes with an additional nonnegative integer, called the *parameter* and traditionally denoted by k, which is the given hardness measure of the instance. A problem is said to be in *class XP* if it admits an algorithm working in $f(k) \cdot n^{f(k)}$ time for some computable function f, where n is the length of the input. A problem is called *fixed-parameter tractable* (FPT, for short), or belongs to *class FPT*, if it admits an algorithm working in $f(k) \cdot n^{\mathcal{O}(1)} = \mathcal{O}^*(f(k))$ time for some computable function f; such running time is also called *fixed-parameter tractable*. These definitions can be naturally extended to allow several parameters by treating k as a vector of nonnegative integers rather than a single number. Since XP algorithms are widely regarded as impractical, we are mostly interested in FPT results. We refer to Section 2.2.1 for more discussion on these notions, including definitions of non-uniform classes XP and FPT.

Parameterized complexity as an algorithm design paradigm was introduced in the beginning of the 90s by Downey and Fellows. The original motivation comes from the Graph Minors project of Robertson and Seymour, whose many algorithmic corollaries may be conveniently re-interpreted as fixed-parameter tractability results. However, it very quickly turned out that the concept is much more general and can be applied to a full variety of fields of computer science, from very theoretical, like the theory of graph minors, to very applied, like bioinformatics or machine learning. Today, parameterized complexity is an established and dynamically developing subbranch of algorithmics with two classical monographs [119, 132], a dedicated annual conference (International Symposium on Parameterized and Exact Computations, IPEC) and a number of workshops organized every

[1] The $\mathcal{O}^*(\cdot)$ notation suppresses factors polynomial in the input size.

year. According to the website of the parameterized complexity community [308], more than 100 research papers related to the paradigm appear each year in peer-reviewed proceedings of international conferences. We refer to the recent festschrift in honor of Mike Fellows [44] for a historical overview of the development of parameterized complexity.

Let us explain the main idea of the parameterized approach on a simple example of the Set Cover problem. In this problem, we are given a universe U, a family $\mathcal{F}$ of subsets of U, and an integer k. The question is whether one can find a subfamily $\mathcal{G} \subseteq \mathcal{F}$ of size at most k such that $\mathcal{G}$ covers every element of U, i.e. $\bigcup \mathcal{G} = U$. Let $|U| = n$ and $|\mathcal{F}| = m$. Observe that we may design two natural algorithms for the problem.

(i) First, we may simply check every possible subset of $\mathcal{F}$ of size k in time $\mathcal{O}^*(\binom{m}{k}) \leq \mathcal{O}^*(2^m)$.

(ii) Second, we may employ a simple dynamic programming routine on subsets of U. Let $F_1, F_2, \ldots, F_m$ be an arbitrary ordering of $\mathcal{F}$. For each subset $X \subseteq U$ and $i = 0, 1, \ldots, m$, let $\phi(X, i)$ denote the minimum possible number of sets among $\{F_1, F_2, \ldots, F_i\}$ that together cover X, or $+\infty$ if no such covering exists. Clearly, for any $X \subseteq U$ we have that $\phi(X, 0)$ is equal to $+\infty$ unless $X = \emptyset$, in which case $\phi(\emptyset, 0) = 0$. On the other hand, the answer to the problem can be found by checking whether the value $\phi(U, m)$ does not exceed k. It is easy to observe, however, that function $\phi(\cdot, \cdot)$ satisfies the following recurrence for $i > 0$ and $X \subseteq U$:

$$\phi(X, i) = \min(\phi(X, i-1), 1 + \phi(X \setminus F_i, i-1)).$$

Therefore, we can compute all the values of the function $\phi(\cdot, \cdot)$ in $\mathcal{O}^*(2^n)$ time and check whether $\phi(U, m) \leq k$, thus obtaining an algorithm working in $\mathcal{O}^*(2^n)$ time.

In the language of parameterized complexity these results imply that the problem is (i) fixed-parameter tractable when parameterized by m, for it admits an $\mathcal{O}^*(2^m)$ algorithm; (ii) fixed-parameter tractable when parameterized by n, for it admits an $\mathcal{O}^*(2^n)$ algorithm; and (iii) in class XP when parameterized by k, for it admits an $\mathcal{O}^*(\binom{m}{k}) \leq \mathcal{O}^*(m^k)$ algorithm. This example shows that even a seemingly innocent and classical problem can hide a rich parameter ecology, which enables us to design two substantially different algorithms that are in some sense complementary: one is essentially faster when the family is smaller, while the other when the universe is smaller. Moreover, two natural questions stem from this example.

Firstly, since we already know that Set Cover is in XP when parameterized by k, can it be in fact fixed-parameter tractable with the same parameterization? Here the answer is (probably) negative. Downey and Fellows defined a hierarchy of complexity classes

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[P] \subseteq XP,$$

called also W-hierarchy; we refer to their monograph [119] for a comprehensive introduction. The hierarchy is conjectured to be strict, that is, every inclusion in the sequence above is proper; this belief is supported by many links connecting collapses in W-hierarchy with collapses in the classical complexity. Set Cover parameterized by k is known to be complete for the class $W[2]$ [119], hence existence of a fixed-parameter tractable algorithm would imply that $FPT = W[2]$.

Secondly, is the constant 2 in the designed algorithms for Set Cover optimal? In other words, can one design algorithms with running times $\mathcal{O}^*((2-\varepsilon)^n)$ or $\mathcal{O}^*((2-\varepsilon)^m)$ for some $\varepsilon > 0$? This question has been investigated by Cygan et al. [86] and the answer is probably negative as well,

but our foundations for making this statement are much less certain. Cygan et al. have shown that an $\mathcal{O}^*((2-\varepsilon)^m)$ algorithm for Set Cover would contradict the *Strong Exponential-Time Hypothesis* (SETH) of Impagliazzo and Paturi [195], which is still broadly assumed, yet considered doubtful by many. For an $\mathcal{O}^*((2-\varepsilon)^n)$ algorithm, Cygan et al. were unable to show links between existence of such an algorithm and SETH; however, they have conjectured that such links exist, and supported this claim by showing certain obstacles for attacking the problem via standard randomized-algebraic techniques. Therefore, the question whether the base of the exponent can be improved for parameterization by n is still open.

The presented example of the Set Cover problem shows that even for very basic problems applying the parameterized complexity paradigm can lead to a large number of concrete and challenging research questions, whose pursuit gives new insights and improves our understanding of the issue. Many of these questions remain open despite simplicity of their formulation.

1.3 Goals of the parameterized analysis

As we have argued, the main goal of the parameterized paradigm is to provide tools for precise, multivariate analysis of NP-hard problems. While the theory of classical complexity uses only one measure of hardness of an instance, namely the total length of the encoding, in parameterized complexity one tries to design additional measures to precisely locate the source of problem's intractability, and understand which properties make a particular input hard to tackle. The thesis is that such a systematic, multivariate analysis of the problem is likely to give new insights that can be used to design algorithms that perform efficiently in practice.

The parameterized complexity framework sets the notion of fixed-parameter tractability as the central concept of efficiency. Recall that a parameterized problem is fixed-parameter tractable if it admits an algorithm working in time $f(k) \cdot n^{\mathcal{O}(1)}$ for some computable function f, where k is the parameter and n is the input length. Given this definition, there are two natural directions of optimizing the running time of a fixed-parameter tractable algorithm:

- First, one can optimize the dependence on the parameter, i.e., make the function f as low as possible.
- Second, one can optimize the polynomial factor, i.e., make the constant exponent of n as small as possible.

For many problems these two goals stand in some sort of a counterposition. In other words, one can optimize the polynomial factor at a cost of worse dependence on the parameter. A classical example of such a situation is the Odd Cycle Transversal problem; here, given a graph G and an integer k, one is asked whether at most k vertices can be deleted from G to make G bipartite. On one hand, a relatively simple FPT algorithm via the technique of iterative compression was proposed by Reed, Smith, and Vetta [277]. The running time of this algorithm is $O(3^k \cdot kmn)$ for graphs with n vertices and m edges[2]. On the other hand, Kawarabayashi and Reed [211] have shown an algorithm that achieves almost linear time in terms of the input size, but at a cost of much worse dependence on k. Kawarabayashi and Reed do not even provide estimates on the function f they obtain. Even though algorithms with better dependence on the parameter than 3^k have been found

[2]Reed et al. claimed only running time $\mathcal{O}(4^k \cdot kmn)$, but a more careful analysis of the algorithm reveals that it may be implemented to run in $\mathcal{O}(3^k \cdot kmn)$ time; cf. [193].

recently [243, 255], it is still open whether one can find an algorithm that both has a subquadratic polynomial factor, and sensible (say, single exponential) dependence on the parameter.

It should be said that different researchers take very different stances with respect to the question whether optimizing the dependence on the parameter or the polynomial factor is more important. For this reason, all the algorithms contained in this thesis have the polynomial factor stated and rigorously analyzed, even though the author's personal viewpoint actually puts more focus on the dependence on the parameter. In fact, the author advocates trying to optimize both factors of the time complexity at the same time whenever possible, as for example in the recent approximation algorithm for treewidth of Bodlaender et al. [46].

1.4 Overview of the techniques

In this section we give a short overview of the main approaches used in parameterized complexity to prove upper and lower bounds; many of the presented concepts will be used later in this thesis. We focus more on approaches understood as general frameworks in which a parameterized problem can be considered, rather than particular techniques or tricks. We refer to the books of Downey and Fellows [119] and of Flum and Grohe [132] for a broader discussion.

1.4.1 Upper bounds

Branching

One can argue that branching algorithms are the salt of parameterized complexity. Consider the classical Vertex Cover problem parameterized by the solution size: given a graph G and an integer k, one is asked whether k vertices of G can be removed so that G becomes edgeless. We perform the following simple strategy: as long as there exist edges in the graph, we pick any edge vw and branch into two subprograms. In the first subprogram we assume that v will be deleted in an optimum solution, and in the second subprogram we assume that w will be deleted. Hence, in each subprogram (also called *branch*) we remove the chosen vertex and continue. We perform this strategy up to the point when there are no edges left in the graph — in which case we report success — or when k vertices have been already removed and there are still some edges left — in which case we declare the branch to be fruitless and terminate it. Observe that if some vertex cover of size at most k exists in the graph, then it will be discovered in at least one branch and thus the algorithm will report success. On the other hand, if there is no vertex cover of size at most k, then of course all the branches will turn out to be fruitless. For the time complexity, observe that since we branched into two possibilities at each step and the search tree has depth bounded by k, then its size is at most $2^{k+1} - 1$. Consequently, the whole algorithm runs in $\mathcal{O}^*(2^k)$ time, thus showing that the Vertex Cover problem is fixed-parameter tractable when parameterized by the requested solution size.

Observe that in order to make sure that a branching algorithm works in FPT time we need to ensure the following two conditions:

(i) at each branching step, the algorithm chooses among a set of choices of cardinality bounded by a function of the parameter;

(ii) the height of the branching tree is bounded by a function of the parameter.

Sometimes we also say that an algorithm *guesses* the correct branch to say that it branches into all the possibilities. This corresponds to viewing an FPT branching algorithm as a polynomial-time

algorithm that can perform some nondeterministic steps, provided that their number is bounded by a function of the parameter.

Branching routines are ubiquitous in the world of parameterized and exact algorithms. On one hand, they provide a simple and independent method of attacking NP-hard problems, which is particularly suitable for optimizing the dependence of the running time on the parameter. For example, for the aforementioned Vertex Cover problem one can achieve as low running time as $\mathcal{O}^*(1.2738^k)$ using extensive branching [69]. On the other hand, simple branching subroutines are also useful in combination with other, more powerful techniques. They are often used as opening steps that simplify the structure of the instance by removing easily resolvable parts, or as finishing steps when the instance is cleaned enough to admit an easy branching algorithm.

Kernelization

Kernelization can be already considered an independent subbranch of parameterized complexity with its own tools and methods. The origin of kernelization is a realization that the parameterized framework can be used to create a rigorous and formal theory that explains the power of preprocessing routines, i.e., polynomial-time algorithms for NP-hard problems that do not solve the problem completely, but shrink the size of the instance at hand as much as possible by identifying parts of the input that are redundant or resolvable in polynomial time, and reducing them. Formally, a *kernelization algorithm* for a parameterized problem L is a polynomial-time algorithm that, given an instance (x, k), outputs an instance (x', k') that is equivalent to the input one, i.e., (x, k) is in L if and only if (x', k') does, and such that $|x'|, k' \leq g(k)$ for some computable function g. The output instance (x', k') is called the *kernel*, while the function g is called the *size of the kernel*. Thus, by applying the parameterized framework we can allow a kernelization algorithm not to solve the problem completely (which we do not expect to happen), but to stop performing simplification steps once the instance is already small enough in terms of the hardness measure of the instance.

A folklore observation states that in fact every decidable problem L is FPT if and only if it admits some kernelization algorithm. Obviously, if a kernelization algorithm for L exists, then we may first apply it, and then run any algorithm resolving L on the obtained kernel; note that this algorithm runs in $f(k)$ time for some computable function f, since the kernel's size is bounded by a computable function of k. To prove the converse implication, assume that L admits an FPT algorithm $\mathbb{A}$ working in time $f(k) \cdot n^{\mathcal{O}(1)}$ and consider the following kernelization algorithm. Given input (x, k), we first check if $|x| \leq f(k)$. If this is the case, then we do not need to do anything, so we output the pair $(x', k') = (x, k)$ as the obtained kernel. On the other hand, if $|x| > f(k)$, then algorithm $\mathbb{A}$ on input (x, k) works in $f(k) \cdot |x|^{\mathcal{O}(1)} \leq |x| \cdot |x|^{\mathcal{O}(1)} = |x|^{\mathcal{O}(1)}$ time, that is, in polynomial time. Hence, we may apply the algorithm $\mathbb{A}$ to (x, k) to solve the instance completely, and as a kernel provide a trivial YES- or NO-instance, depending on the result.

This observation suggests that the general notion of kernelization is basically the same as fixed-parameter tractability. However, restricting our attention to *polynomial kernels*, that is, requiring that the size of the kernel is a polynomial function of the parameter, results in a much richer landscape of results. Note that polynomial kernels are particularly interesting, as applying any brute-force algorithm on a polynomial kernel yields a very efficient FPT algorithm for a problem. Many classical combinatorial problems indeed do possess polynomial kernels, with notable examples of Vertex Cover [68] and Feedback Vertex Set [303]. The interest in polynomial kernels has spiked since the discovery of the so-called composition framework, which provides tools for proving that certain problems do not admit polynomial kernels unless NP $\subseteq$ coNP/poly [45, 120, 150].

Examples of problems without polynomial kernels are k-Path [45] and Set Cover for both FPT parameterizations: by n or by m [116].

The study of kernelization algorithms have brought many new tools and techniques into the world of fixed-parameter tractability. Of particular significance are meta-theorems for obtaining linear kernels for a class of problems on graphs with topological constraints, developed by Fomin et al. [49], and the novel technique of Kratsch and Wahlström of obtaining randomized kernels via the use of matroids [225, 226]. On the other hand, from the practitioner's point of view kernelization algorithms are free of the main weak point of parameterized complexity, namely that the definition of fixed-parameter tractability inherently allows exponential-time computations. Thus, the kernelization framework has found many applications, among others in computational biology. In modern parameterized complexity, investigating possibilities of polynomial kernelization is usually an immediate follow-up question after proving that a problem is fixed-parameter tractable. We refer to a recent survey of Lokshtanov et al. [242] for a more comprehensive introduction to kernelization.

Dynamic programming and width measures

As we have seen in the example of Set Cover, dynamic programming routines can be very useful for designing FPT algorithms. Similarly as in this example, one often employs a dynamic program on a family of states that is exponential, but bounded by a function of the parameter. Usually states are formed simply by subsets of some small universe, but more complicated families of states can also be used; we will see many such examples in this thesis. Let us remark that similar usage of dynamic programming is ubiquitous in the world of exact algorithms.

In parameterized complexity a standard approach is to design dynamic programming routines on *structural decompositions*. The idea is as follows: even though the graph itself can be large, maybe its structure can be described by some sort of a simple decomposition on which a dynamic program can be employed. Simplicity of the decomposition is usually expressed by its parameter called the *width measure*. Consider for example the Independent Set and Dominating Set problems. While NP-hard in general, both of the problems can be solved in linear time on trees by a simple dynamic programming routine. Therefore, it is natural to ask whether there exists a notion of 'tree-likeness' that measures how much a graph resembles a tree, such that the simple dynamic program for trees can be lifted also to graphs with small value of this 'tree-likeness'.

And indeed, this intuitive 'tree-likeness' has been formalized independently by several researchers in the 80s; the version used nowadays, called *treewidth*, was introduced by Robertson and Seymour in their seminal Graph Minors project [280]. The formal layer of this notion is somewhat technical (we provide details in Section 2.3.2; see also the book of Diestel [112]), but for the purpose of this overview let us imagine that a graph of treewidth t resembles a tree where the width of the trunk is not 1 as in the classical definition of a tree, but t instead.

Treewidth is today an important tool for a number of fields of theoretical computer science, including artificial intelligence, approximation, computational biology, and, most importantly for us, parameterized complexity. The reason is that for many NP-hard problems one can implement very efficient dynamic programming routines working on graphs of small treewidth; for instance, the Independent Set and Dominating Set problems can be solved in times $\mathcal{O}(2^t \cdot t^{\mathcal{O}(1)} \cdot n)$ and $\mathcal{O}(3^t \cdot t^{\mathcal{O}(1)} \cdot n)$ on graphs of treewidth t, respectively. On the other hand, graphs of small treewidth actually do appear in applications. For instance, via the classical Lipton-Tarjan planar separator theorem [238] it can be easily shown that a planar graph on n vertices has treewidth at most $\mathcal{O}(\sqrt{n})$, which immediately yields $2^{\mathcal{O}(\sqrt{n})}$ algorithms for the Independent Set and Dominating

SET problems on planar graphs. Dynamic programming algorithms for graphs of bounded treewidth can be used as a method on its own, but most often are applied as subroutines in combination with other techniques.

The concept of treewidth also seems to uncover a fundamental link between the topological complexity of graphs, measuring which was the original motivation of this notion, and the complexity of model checking natural models of logic on graphs. Courcelle observed that the classical equivalence of the *Monadic Second-Order* (**MSO**) logic on finite trees with tree automata can be lifted to graphs of small treewidth, thus discovering a powerful meta-theorem for designing algorithms on graphs of bounded treewidth, today known as the *Courcelle's Theorem* [79]. This theorem can be stated as follows: there exists an algorithm that, given a formula φ of $\mathbf{MSO}_2$ logic on graphs and a graph G of treewidth at most t, checks whether φ is satisfied in G in time $f(||\varphi||, t) \cdot n$ for some computable function f. Here, $\mathbf{MSO}_2$ means that we allow quantification both over subsets of vertices and over subsets of edges; see Section 2.3.6 for a formal definition. In other words, Courcelle's theorem states that model checking $\mathbf{MSO}_2$ on graphs is fixed-parameter tractable when parameterized jointly by the length of a formula and the treewidth of a graph. Thus, existence of many efficient dynamic programming routines on graphs of small treewidth can be explained by a common reason: expressibility in $\mathbf{MSO}_2$.

Following the success of treewidth, also other width notions have been introduced and are used. *Cliquewidth*, for instance, has been introduced by Courcelle and Olariu [83] to measure the complexity of a graph viewed as an algebraic structure, rather than a topological space as was the case for treewidth. Similarly to treewidth, also graphs of bounded cliquewidth admit efficient dynamic programming routines, and an analogue of Courcelle's theorem can be proved if one forbids quantification over sets of edges [82]. We invite the reader to Chapter 3 for a more thorough introduction to width measures and dynamic programming algorithms on graph decompositions.

WIN/WIN approaches

The WIN/WIN approach is an intuitive algorithm design paradigm that can be stated as follows. Given the input to the program, we perform some preliminary check. Depending on the outcome, we choose different strategies of tackling the problem, exploiting different features of the input that the check exhibited. The name of the technique comes from the fact that each possible outcome of the check should provide us with information that enables us to solve the problem efficiently; in informal words, for every possible result of the check we win.

In parameterized complexity WIN/WIN approaches are most often used in combination with dynamic programming on graph decompositions. The preliminary check is usually an attempt to construct a simple decomposition of the input on which a fast dynamic program can be employed. If we succeed in constructing the decomposition, then we win by applying the dynamic programming algorithm and solving the problem quickly. Otherwise, there must be some reason of construction's failure. This reason usually takes form of a complicated combinatorial object embedded in the graph, whose complexity prevents us from decomposing it expeditiously; such an object is usually referred to as an *obstacle*. Informally speaking, while constructing the decomposition we may discover a big, entangled knot that we are not able to handle. The crucial insight of the WIN/WIN approach is that such an obstacle can be also exploited algorithmically. Usually, we are either able to immediately provide an answer to the problem by examining the complicated structure of the obstacle, or find a part of the obstacle that is provably irrelevant to the problem and can be safely removed (design a so-called *irrelevant vertex rule* or *irrelevant edge rule*). After this removal we

restart the whole algorithm.

The WIN/WIN approach in parameterized complexity is most often used in the context of treewidth and tree decompositions. The reason is that the obstacle for having small treewidth is a large *grid minor*: an extremely useful and natural combinatorial object that can be exploited in many different ways. In Section 3.2.4 we survey algorithmic results that can be obtained via WIN/WIN approaches that use grid minors.

Well-quasi-orderings

The well-quasi-ordering framework is perhaps one of the oldest algorithmic techniques used in parameterized complexity, and stems from the original motivation of the field, that is, the Graph Minors project of Robertson and Seymour. Formally, a partial order $(P, \leq)$ is a *well-quasi-ordering* (WQO, for short) if for any infinite sequence of elements $x_1, x_2, x_3, \ldots \in P$ there exist indices $i < j$ such that $x_i \leq x_j$. A folklore observation states that this is equivalent to the following definition. Let $S \subseteq P$ be any subset of P. We say that S is *closed under* $\leq$ if whenever $x \leq y$ and $y \in S$, then also $x \in S$. We say that S is *characterized by obstruction set* $\mathcal{F}_S$ if $S = \{x \in P \mid \neg\exists_{f \in \mathcal{F}_S}\ f \leq x\}$; in other words, S comprises all the elements of P that are not larger or equal to any element of the obstruction set. Then $(P, \leq)$ is a well-quasi-ordering if and only if it does not contain any infinite decreasing sequences and *every* set S closed under $\leq$ is characterized by a *finite* obstruction set.

Let us now explain the well-quasi-ordering framework on the example of the Graph Minors theorem of Robertson and Seymour [287], which states that the class of undirected graphs ordered by the minor ordering (see Section 2.3.1 for the definition of a minor) is a well-quasi-ordering. In other words, every class of graphs that is closed under taking minors is characterized by a *finite* set of forbidden minors, depending on the class only. For instance, the classical results of Kuratowski [234] and of Wagner [307] prove that the class of planar graphs is characterized by the set of forbidden minors $\{K_{3,3}, K_5\}$. More generally, for any 2-dimensional manifold M, the class of graphs that can be embedded on M without crossings is closed under taking minors, so it can be characterized by a finite set of forbidden minors $\mathcal{F}_M$. Note, however, that we can prove only *existence* of set $\mathcal{F}_M$, and the Graph Minors theorem of Robertson and Seymour does not provide us any means of computing $\mathcal{F}_M$, or even bounding its size. In fact, $\mathcal{F}_M$ is not known even for M being a torus, and the only complete classification besides the plane has been done for the projective plane [254] (the family of forbidden minors is of size 35).

Besides the Graph Minors theorem, the theory of Robertson and Seymour has also many important algorithmic corollaries. In particular, Robertson and Seymour [285] have proven that checking whether a graph H is a minor of G can be done in time $f(|V(H)|) \cdot n^3$, where $n = |V(G)|$ and f is some computable function. In other words, minor testing is fixed-parameter tractable when parameterized by the size of the tested minor. When we combine this algorithm with the Graph Minors theorem, we obtain the following surprisingly strong corollary: for every class of graphs S closed under taking minors, membership in S can be tested in $c_S \cdot n^3$ time for a constant c_S depending on S only. We namely just need to test whether any graph from $\mathcal{F}_S$ is a minor of the input graph. This provides a powerful algorithmic meta-tool for determining whether a problem is (non-uniformly) fixed-parameter tractable.

Let us explain this technique on the example of treewidth. It can be easily seen that the class $\mathcal{F}_t$ of graphs of treewidth at most t is closed under minors; we also say that treewidth is a graph parameter *closed under taking minors*. Hence, for every t there exists an algorithm working in time $c_t \cdot n^3$ checking whether an n-vertex graph has treewidth at most t. Note that according to our definition

of fixed-parameter tractability, this does *not* mean that computing treewidth is fixed-parameter tractable when parameterized by t, since we have designed a separate algorithm for every value of t, and not a single algorithm that takes t as input. In fact, each of the designed algorithms has the corresponding family $\mathcal{F}_t$ hard-coded in its code; as we have noted, the Graph Minors theorem does not provide us any method of deriving $\mathcal{F}_t$ knowing the value of t only. Such algorithms, or rather families of algorithms, are called *non-uniform FPT algorithms* (see Section 2.2.1 for formal details).

Similarly to treewidth, also many other graph parameters are closed under taking minors, and hence their value can be computed by non-uniform FPT algorithms; examples include vertex cover number, feedback vertex set number and genus, among many others. However, such algorithm are clearly completely impractical. It is not only that we cannot control the running of the algorithm; we in fact cannot even implement it, as the implementation depends on the hard-coded obstruction set that is unknown. The only information that the Graph Minors theorem provides us is a non-constructive proof that such an algorithm *exists*. For this reason, well-quasi orderings are most often used only as preliminary classification tools, whose usage should be always followed by investigation of existence of a uniform FPT algorithm, with running time as low as possible.

1.4.2 Lower bounds

W-hardness

Since it is commonly believed that $FPT \neq W[1]$, proving that a problem is $W[t]$-hard for some $t \geq 1$ shows that the existence of an FPT algorithm is unlikely. As usual, $W[t]$-hardness of a parameterized problem L can be proven by providing an appropriate reduction from some known $W[t]$-hard problem L'. In this setting, we use *parameterized reductions*, i.e., reductions that work in FPT time, and which produce an instance whose parameter is bounded by a computable function of the input parameter. Note that then the existence of an FPT algorithm for L together with a parameterized reduction from L' to L would give an FPT algorithm for L'. In the literature $W[1]$- and $W[2]$-hardness results are most common. The reason is that two natural parameterized problems, Clique parameterized by the target clique size, and Dominating Set parameterized by the target dominating set size, are $W[1]$- and $W[2]$-complete [119], respectively, thus providing a very convenient source of reductions. Since the formal definition of classes $W[t]$ is somewhat technical and irrelevant with respect to the content of this thesis, we omit it and refer to the monograph of Downey and Fellows [119] for a more thorough introduction to the W-hierarchy. For our purposes, it suffices to think of $W[1]$ as the class of problems reducible to Clique via parameterized reductions, and of $W[2]$ as the class of problems reducible to Dominating Set via parameterized reductions.

Constructing $W[1]$- and $W[2]$-hardness reductions is somewhat similar to classical NP-hardness reductions, but of course the character of the goal is different. Intuitively, the gist of an NP-hardness reduction, say from the SAT problem, is to create a uniform search space of n binary choices that can be checked with each other. For a $W[1]$-hardness reduction, say from the Clique problem, one usually needs to express a search space of size n^k corresponding to possible choices of k-tuples of vertices that are candidates for the clique. Hence, while a usual gadget in an NP-hardness reduction is of constant size and expresses a *constant* number of states, in a $W[1]$-hardness reduction a common opening step is to construct a gadget that expresses an *unbounded* number of states, the so-called 1-*in-n gadget*. Similarly as with NP-hardness, an experienced researcher working in parameterized complexity can usually easily identify features of a problem that can be used for designing a $W[1]$-hardness reduction; these features can be intuitively described as the ability to

express k independent 1-in-n choices that can be pairwise checked with each other.

ETH and SETH

$W[t]$-hardness reductions can be used to identify a problem's location on the complexity landscape only very crudely. We can distinguish FPT problems from problems that are not likely to be fixed-parameter tractable. However, for example distinguishing FPT problems that require non-elementary dependence on the parameter from problems that are solvable in subexponential parameterized time, i.e., time $\mathcal{O}^*(2^{o(k)})$, using W-hardness reductions only seems implausible. Similarly, also for problems in XP that are not likely to be in FPT it is interesting to ask whether the exponent of the polynomial factor needs to be exponential in the parameter, or for instance can be sublinear. The *Exponential-Time Hypothesis* and *Strong Exponential-Time Hypothesis* of Impagliazzo and Paturi [195] are assumptions using which we can make more precise statements about lower bounds on time complexities of parameterized algorithms.

ETH and SETH are based on hardness of the CNF-SAT problem. Recall that in the CNF-SAT problem we are given a propositional formula φ over a set of n boolean variables, and we assume that φ is in *conjunctive normal form*. That is, φ is a conjunction of m clauses, each being a disjunction of a number of literals — variables appearing in negated or non-negated form. The question is whether there exists an evaluation of the variables to values true or false such that φ is satisfied. In the q-CNF-SAT problem we additionally require that every clause contains at most q literals. While 2-CNF-SAT problem is polynomial-time solvable [18], already 3-CNF-SAT is a classical NP-hard problem.

Observe that the CNF-SAT problem can be trivially solved in $\mathcal{O}^*(2^n)$ time by trying all the possible evaluations. However, for every $q \geq 3$ there exists $\varepsilon_q < 1$ such that q-CNF-SAT can be solved in $\mathcal{O}^*(2^{\varepsilon_q n})$ time[3] [266]. For the case of $q = 3$, ε_3 is known to be as low as 0.386. However, for the currently known algorithms it holds that $\lim_{q\to\infty} \varepsilon_q = 1$, that is, as we relax the constraint on the length of the clauses, the brute-force search seems more and more inevitable. Therefore, let us denote by δ_q the infimum over the set of constants c for which an $\mathcal{O}^*(2^{cn})$ algorithm solving q-CNF-SAT exists, for $q \geq 3$. In this definition we allow randomized two-sided error algorithms.

The *Exponential-Time Hypothesis* (ETH) states that $\delta_3 > 0$. In other words, there exists a constant $c > 0$ such that no algorithm for 3-CNF-SAT can achieve running time $\mathcal{O}^*(2^{cn})$. The *Strong Exponential-Time Hypothesis* (SETH) states that $\lim_{q\to\infty} \delta_q = 1$, that is, that brute-force search becomes necessary as q tends to infinity. Note that in particular, ETH implies that no $\mathcal{O}^*(2^{o(n)})$ exists for 3-CNF-SAT, while SETH implies that no $\mathcal{O}^*((2-\varepsilon)^n)$ exists for the general CNF-SAT problem, for any $\varepsilon > 0$. ETH is most often used in combination with *Sparsification Lemma* of Impagliazzo et al. [196], which intuitively states that for the sake of designing exponential-time algorithms for q-CNF-SAT one can assume that the number of clauses is linear in the number of variables, where the ratio between the number of clauses and the number of variables depends on q. The immediate consequence of Sparsification Lemma is that when defining ETH, one can replace n with $n + m$ in the definitions of constants δ_q. In other words, ETH is equivalent to claiming that there exists a constant $c > 0$ such that no algorithm for 3-CNF-SAT can achieve running time $\mathcal{O}^*(2^{c(n+m)})$. We refer to Section 2.2.3 for a formal introduction of ETH, SETH, and Sparsification Lemma.

It should be noted that while ETH is generally considered a plausible complexity assumption, SETH is regarded by many as a quite doubtful working hypothesis that can be refuted any time.

[3] Here, the polynomial factor hidden in the $\mathcal{O}^*(\cdot)$ notation may depend on q.

For this reason, lower bounds proven under the assumption of SETH should not be regarded as supported by very strong arguments, but rather that existence of better algorithms would constitute a major breakthrough in the complexity of satisfiability.

ETH is a very convenient source of reductions for proving precise statements about running times of exact and parameterized algorithms. For instance, in order to show that an NP-hard problem is not solvable in subexponential time in terms of the input length, it suffices to provide a reduction from 3-CNF-SAT that produces an instance of size linear in the size of the input formula. Pipelining such a reduction with the assumed subexponential-time algorithm would yield a subexponential-time algorithm for 3-CNF-SAT, contradicting ETH. Similarly, to refute the existence of a subexponential parameterized algorithm, i.e., an FPT algorithm working in time $\mathcal{O}^*(2^{o(k)})$, it suffices to present a reduction from 3-CNF-SAT to a parameterized problem that produces an instance whose parameter is bounded linearly in the size of the input formula. As Flum and Grohe observe [132], most of the known reductions for classical parameterized problems have in fact the aforementioned property, and thus the corresponding problems do not admit subexponential parameterized algorithms unless ETH fails. However, ETH is a much more general tool for proving tight bounds on time complexity. It has been used for example to prove optimality of some slightly super-exponential parameterized algorithms [241], of classical dynamic programming on graphs of bounded treewidth [91, 241], or of XP algorithms for problems known to be $W[1]$-hard [64, 67, 133, 251], as well as to sketch the limits of applicability of Courcelle's theorem [228, 231, 232]

While ETH can be used to estimate the order of the exponent of the running time of a parameterized algorithm, SETH is an even more precise tool using which one can pinpoint tight bounds on the base of the exponent. Contrary to ETH reductions, in SETH reductions one usually needs to start with an arbitrary q-CNF-SAT instance instead of simple 3-CNF-SAT, and very carefully control the parameter of the constructed output instance instead of just ensuring linear dependence on the size of the input formula. Similarly to ETH, also SETH can be used for a large variety of parameterized problems. For example, Lokshtanov et al. [239] and Cygan et al. [91] have proven a number of lower bounds on the classical dynamic programs for graphs of bounded treewidth. In particular, Lokshtanov et al. have proven that for any $\varepsilon > 0$, existence of algorithms solving the Independent Set and Dominating Set problems on graphs of treewidth t and achieving running times $\mathcal{O}^*((2-\varepsilon)^t)$ and $\mathcal{O}^*((3-\varepsilon)^t)$, respectively, would contradict SETH. On the other hand, Cygan et al. [86] attempted to create a net of SETH reductions between many important problems of parameterized complexity. Despite not linking all the problems to SETH, the landscape given in [86] provides a fairly good understanding of hidden links between fundamental problems for which brute-force or straightforward dynamic programming solutions seem to be optimal. It is noteworthy that SETH can be also used to prove refined lower bounds for the performance of some XP and polynomial-time algorithms [272].

In Chapter 4 we provide a more thorough survey of the program of obtaining tight bounds on the complexity of parameterized problems.

Kernelization lower bounds

In the overview of the field of kernelization we briefly mentioned the methodology for proving implausibility of polynomial kernelization. Currently, the main tool for showing that a parameterized problem probably does not admit a polynomial kernel is the composition framework of Bodlaender et al. [45], with backbone information-theoretical theorem proved by Fortnow and Santhanam [150]. Bodlaender et al. have shown that a parameterized problem does not admit a polynomial kernel

unless NP $\subseteq$ coNP/poly providing it is *OR-compositional*. Intuitively, OR-compositionality means that given a large number of instances of the problem with the same parameter k, one can merge these instances in polynomial time into one, possibly huge instance, but whose parameter is bounded by a polynomial of k. The output instance should be equivalent to logical OR of the input instances. We provide formal details of the composition framework in Section 2.2.4.

A simple example of compositionality is the k-Path problem, where, given a graph G and integer k, one is asked if a simple path of length k exists in G. For the merging procedure, the so-called *composition algorithm*, we simply take the disjoint union of input instances, and ask for the same value of k. Observe that a k-path exists in the disjoint union of graphs if and only if it exists in any of them; hence the output instance is equivalent to the logical OR of the input instances. Therefore, by a simple application of the results of Bodlaender et al. we obtain that the k-Path problem does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly.

The intuition behind compositionality is as follows. Assume that a compositional problem admits a polynomial kernel. Then, by pipelining the composition algorithm with kernelization we are able to pack information contained in an *unbounded* number of input instances into an instance of size only polynomial in terms of the parameter. Thus, some information about some of the instances must have been lost during compression into the kernel. However, we do not expect a polynomial-time algorithm to be able to identify the instances on the input that can be safely discarded. This is precisely the intuition behind the backbone theorem of Fortnow and Santhanam [150] that led to development of the composition framework.

Since its discovery, the composition framework has been used several times to refute existence of polynomial kernels for many different problems. It should be said that it is rare that a composition algorithm is so simple as in the case of the k-Path problem. Usually, designing a composition algorithm requires both a good understanding of the problem, and a number of combinatorial ideas. Today, most compositions use the notion of *cross-composition*, introduced by Bodlaender et al. [50], that puts the original framework into a very convenient and useful formalism.

Dell and van Melkebeek [102] have refined the backbone result of Fortnow and Santhanam to provide a framework for proving more precise lower bounds on the sizes of kernels. For instance, they have proved that the Vertex Cover problem cannot admit a kernelization algorithm that compresses any instance into bitsize $\mathcal{O}(k^{2-\varepsilon})$ for any $\varepsilon > 0$, unless NP $\subseteq$ coNP/poly. The framework, today known as *weak compositions*, has been further improved and applied to other problems by Dell and Marx [101] and by Hermelin and Wu [188].

It should be also mentioned that Bodlaender et al. [45] have conjectured that the OR function in OR-compositions can be replaced by the AND function. This statement, further known as the *AND-conjecture*, has been recently resolved positively by Drucker [120]. That is, Drucker has shown that existence of a polynomial kernel for an AND-compositional problem also contradicts the assumption that NP $\not\subseteq$ coNP/poly. The positive resolution of the AND-conjecture refutes existence of polynomial kernels for a number of important problems, including for instance computing the treewidth of a graph.

We again invite an interested reader to the survey of Lokshtanov et al. [242] for more information on kernelization lower bounds.

1.5 Highlights of this thesis

1.5.1 Survey of the content

The content of this thesis is based on the following 5 papers, and in particular it contains verbatim material taken from these works.

- *Jungles, bundles, and fixed-parameter tractability*, co-authored by Fedor V. Fomin, and presented at the 24th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013 [146];
- *Computing cutwidth and pathwidth of semi-complete digraphs via degree orderings*, presented at the 30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, Kiel, Germany, February 27 - March 2, 2013 [269];
- *Subexponential parameterized algorithm for computing the cutwidth of a semi-complete digraph*, co-authored by Fedor V. Fomin, and to be presented at the 19th Annual European Symposium, ESA 2013, Sophia Antipolis, France, September 2-4, 2013 [147];
- *Tight bounds for parameterized complexity of* Cluster Editing, co-authored by Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk and Yngve Villanger, and presented at the 30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, Kiel, Germany, February 27 - March 2, 2013 [137];
- *Known algorithms for* Edge Clique Cover *are probably optimal*, co-authored by Marek Cygan and Marcin Pilipczuk, and presented at the 24th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013 [94].

The thesis consists of two parts. The first part is based on papers [146, 269, 147] and treats of construction of an algorithmic containment theory for tournaments, or more generally, semi-complete digraphs (see Section 2.1.2 or Section 5.1 for a precise definition of this generalization). It appears that in spite of serious obstacles for lifting the results of the Graph Minors project to the world of general directed graphs, a similar theory can be designed for the class of semi-complete digraphs. This theory has been recently developed by Chudnovsky, Fradkin, Kim, Scott, and Seymour in a series of papers [72, 73, 74, 152, 153, 216]. The work of Chudnovsky, Fradkin, Kim, Scott, and Seymour answers many important questions, but a number of issues, mostly related to algorithmic aspects of the theory, have been left open. In this thesis we address these issues by providing efficient algorithms for most important topological problems in semi-complete digraphs, mirroring the achievements of the Graph Minors project of Robertson and Seymour, of course on a much smaller scale. In particular, we provide new, unified proofs of many of the fundamental findings of Chudnovsky, Fradkin, Kim, Scott, and Seymour, which also lead to more efficient algorithms.

In this part of the thesis we mostly rely on defining width measures of graphs, designing dynamic programming on width decompositions, and applying different WIN/WIN approaches. To give more background on our reasonings, in Chapter 3 we survey applications of width measures in parameterized complexity.

The second part of the thesis addresses the optimality program, that is, pursuit of tight bounds on the complexity of parameterized problems.

First, in Chapter 8 we perform a tight multivariate parameter analysis of the Cluster Editing problem. Shortly speaking, the Cluster Editing problem treats of transforming a graph

into a *cluster graph*, i.e., a disjoint union of cliques (called *clusters*) using a small number of edge additions or deletions. We address two natural parameters of the problem: the allowed number of editions k and the target number of clusters p. Our study reveals a fully tight picture of the interplay between these two parameters. On one hand, we show that the problem can be solved in $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{pk})})$ time. On the other hand, for every magnitude of p being a function of k we show that no algorithm with running time $\mathcal{O}^*(2^{o(\sqrt{pk})})$ exists unless ETH fails. To the best of author's knowledge, this is the first parameterized problem for which such a tight multivariate analysis has been performed. This chapter is based on paper [137].

Second, in Chapter 9 we address the Edge Clique Cover problem, which asks whether the edges of the input graph G can be covered using at most k cliques in G. It was known that the problem admits a simple kernel with at most 2^k vertices, and application of a simple dynamic programming routine on this kernel yields an FPT algorithm with running time $\mathcal{O}^*(2^{2^{\mathcal{O}(k)}})$. We prove a somewhat surprising result that these results are tight under ETH. More precisely, we provide a reduction from the 3-CNF-SAT problem that at the same time shows (i) non-existence of a subexponential kernel, under P$\neq$NP; (ii) non-existence of an algorithm achieving running time $\mathcal{O}^*(2^{2^{o(k)}})$, under ETH. To the best of author's knowledge, this is the first doubly-exponential lower bound on the running time for a natural parameterized problem, and one of a few kernelization lower bounds outside the framework of composition. This chapter is based on paper [94].

In this part of the thesis we address the optimality program of investigating tight bounds for parameterized problems. Again, we survey the advances of the optimality program in Chapter 4 to put our study in a larger context.

1.5.2 Sparse graphs and dense graphs

We would like to point out here that in all the results contained in this thesis we address classes of graphs that are *dense*. The underlying undirected graph of a tournament is a clique, and the combinatorial richness of this graph class is obtained only by choosing different directions of arcs. In the Cluster Editing problem we aim at transforming the given graph into a small number of disjoint cliques using a small number of editions. Hence, we may assume that the input graph is already dense, as it has to be dense after a slight perturbation. A YES-instance of the Edge Clique Cover problem also needs to be dense, since its edge set can be covered by only a few cliques.

As we will try to convince the reader about in Chapter 3, there is a full variety of tools for proving fixed-parameter tractability results on classes of graphs that are sparse, but only a handful of means of using density. However, there are many examples where dense instances appear naturally in applications; see the introductory sections of Chapters 8 and 9 for motivation of Cluster Editing and Edge Clique Cover. A simple example when the input graph cannot be assumed to be sparse, is when it represents some similarity relation between a set of objects. In such a situation the graph is likely to contain large cliques or clique-like clusters of objects of the same type. Understanding possibilities of designing fast algorithms for such graphs is certainly an important and interesting issue.

The common theme of all the reasonings contained in this thesis is that we exploit the dense nature of the input instances to design fast algorithm, or to prove high lower bounds. Even though we do not claim to create any algorithmic theory for dense graphs, we hope that the insights of this thesis will be helpful in future study of the parameterized complexity in dense graph classes.

Chapter 2

Preliminaries

2.1 Notation

2.1.1 General notation

By $\mathbb{N}$ we denote the set of nonnegative integers. For $t \in \mathbb{R}$, we denote $\exp(t) = e^t$ where e is the base of natural logarithms. A *partition* of a set X is a sequence of pairwise disjoint subsets of X whose union is equal to X. By $\triangle$ we denote the symmetric difference of two sets, i.e., $A \triangle B = (A \setminus B) \cup (B \setminus A)$. For a formula of any logic φ, by $||\varphi||$ we denote the length of φ in any fixed encoding. We also use Iverson notation: for a condition ψ, $[\psi]$ denotes value 1 if ψ is true and 0 otherwise.

Let us define function $\mathbf{tower}\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ recursively as follows: $\mathbf{tower}(0, k) = k$ and $\mathbf{tower}(q+1, k) = 2^{\mathbf{tower}(q,k)}$. Following Frick and Grohe [155], a function $f\colon \mathbb{N} \to \mathbb{N}$ is called *elementary* if it can be formed from the successor function, addition, subtraction, and multiplication using finitely many compositions, projections, bounded additions and bounded multiplications (of the form $\sum_{z \leq y} g(\bar{x}, z)$ and $\prod_{z \leq y} g(\bar{x}, z)$). Observe that if $f(k)$ is elementary, then for some constant $q \in \mathbb{N}$ it holds that $f(k) \leq \mathbf{tower}(q, k)$ for large enough k; cf. [85].

The $\mathcal{O}^*(\cdot)$ notation hides factors polynomial in the input size. That is, a function $g(k, |x|)$ is in class $\mathcal{O}^*(f(k))$ if and only if $g(k, |x|) = \mathcal{O}(f(k) \cdot |x|^c)$ for some constant c. Here, k can be an integer parameter, as well as a vector of parameters.

2.1.2 Notation for graphs

Basic definitions

For graph notation, we mostly follow Diestel [112]. An *undirected graph* is a pair $G = (V, E)$, where V, the *vertex set*, is any finite set, and E, the *edge set*, is a family of two-element subsets of V, called the *edges*. An edge between vertices v and w is denoted by vw. We also say that vw is *incident* with vertices v and w, and that v and w are *adjacent* or that they are *neighbors*. Following these definitions, all the graphs contained in this thesis are *simple* unless explicitely stated. That is, we do not allow

- *loops*: one-element subsets in E corresponding to edges connecting a vertex with itself;
- *multiple edges*: multiple copies of the same subset in E corresponding to many edges connecting the same vertices.

By allowing loops and multiple edges in the definition of a graph we arrive at the notion of a *multigraph*.

By $V(G)$ and $E(G)$ we denote the vertex and the edge set of a (multi)graph, respectively. For $X, Y \subseteq V(G)$, we denote $E(X,Y) = \{vw \in E(G) \mid v \in X \wedge w \in Y\}$; we use $E_G(\cdot,\cdot)$ instead of $E(\cdot,\cdot)$ whenever G is not clear from the context. The *size* of a (multi)graph G, denoted $|G|$, is defined as $|G| = |V(G)| + |E(G)|$. For undirected graphs G, H with $V(G) = V(H)$, by $\mathcal{H}(G,H)$ we denote the number of edge modifications needed to obtain H from G, i.e., $\mathcal{H}(G,H) = |E(G) \triangle E(H)|$. We say that two graphs G, H are *isomorphic*, denoted $G \cong H$, if there exists a bijection $\eta\colon V(G) \to V(H)$ such that $uv \in E(G)$ if and only if $\eta(u)\eta(v) \in E(H)$ for all $(u,v) \in E(G) \times E(G)$.

A digraph, or a directed graph, is a pair $G = (V, E)$, where V is again a finite vertex set, while E, the *arc set*, is a family of ordered pairs of different elements of V, called the *arcs*. We refer to a pair $a = (v,w) \in E$ as to an arc *directed* from v to w. We also call w the *head* of a, and v the *tail* of a. Again, all the digraphs contained in this thesis are *simple* unless explicitely stated, that is, they do not contain loops or multiple arcs, defined similarly as in the undirected case. Note, however, that for two different vertices v, w we allow existence of both arcs (v,w) and (w,v) simultaneously. By allowing loops and multiple edges we arrive at the notion of a *multidigraph*.

Definitions of $V(G)$, $E(G)$, and $|G|$ can be extended naturally to the directed setting. Similarly, we also denote $E(X,Y) = \{(v,w) \in E(G) \mid v \in X \wedge w \in Y\}$.

If H and G are undirected graphs, then we say that H is a *subgraph* of G, denoted $H \subseteq G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For an undirected graph $G = (V, E)$ and $X \subseteq V$, by *subgraph induced by* X we mean graph $G[X] = (X, E(X,X))$. The notions of a subgraph and of an induced subgraph can be naturally extended to multigraphs and to digraphs; in the latter case, we talk about *subdigraphs*.

For a graph G, a vertex set $X \subseteq V(G)$ and an edge set $F \subseteq E(G)$, by $G \setminus X$ and $G \setminus F$ we denote graphs $G[V(G) \setminus X]$ and $(V(G), E(G) \setminus F)$, respectively. We also say that G is obtained by *removing* X or F from the graph. Again, these notions can be naturally generalized to multigraphs and digraphs.

A *clique* in an undirected graph G is a subgraph of G where every two vertices are adjacent. We often identify a clique with its vertex set, and by the *size* of a clique we mean the cardinality of its vertex set. Similarly, an *independent set* is a set of vertices that are pairwise non-adjacent. A *vertex cover* is a set of vertices that contains at least one endpoint of every edge of the graph; equivalently, removing a vertex cover from the graph results in an edgeless graph. A *dominating set* is a set of vertices X such that any vertex of the graph either belongs to X or has a neighbor in X. Parameterized problems Clique, Independent Set, Vertex Cover, Dominating Set are defined with respect to the natural parameters being the target cardinalities of the corresponding sets. That is, in all these problems we are given a graph G and an integer k; in Clique and Independent Set we ask whether there exists a clique/independent set on at least k vertices, while in Vertex Cover and Dominating Set the question is whether there exists a vertex cover/dominating set of size at most k.

Neighborhoods and degrees

For an undirected graph G and a vertex $v \in V(G)$, the *open neighborhood* of v is defined as $N(v) = \{w \in V(G) \mid vw \in E(G)\}$, and the *closed neighborhood* of v is defined as $N[v] = N(v) \cup \{v\}$. We extend this notion to subsets $X \subseteq V(G)$ as follows: $N[X] = \bigcup_{v \in X} N[v]$ and $N(X) = N[X] \setminus X$. The *degree* of a vertex v is defined as $d(v) = |N(v)|$.

For a directed graph G, the *open in- and outneighborhoods* of v are defined as $N^-(v) = \{w \in V(G) \mid (w,v) \in E(G)\}$ and $N^+(v) = \{w \in V(G) \mid (v,w) \in E(G)\}$. Similarly as for the undirected graphs, we define closed in- and outneighborhoods and extend these notions to vertex subsets as follows: $N^r[v] = N^r(v) \cup \{v\}$, $N^r[X] = \bigcup_{v \in X} N^r[v]$ and $N^r(X) = N^r[X] \setminus X$, where $r \in \{-,+\}$. The *indegree* of a vertex v is defined as $d^-(v) = |N^-(v)|$, while the *outdegree* is defined as $d^+(v) = |N^+(v)|$. The *degree* of v is defined as $d(v) = d^-(v) + d^+(v)$.

Paths and cycles

Let G be an undirected graph. A *walk* in G is a sequence of vertices such that every two consecutive vertices of the sequence are connected by an edge. The *length* of a walk is equal to the length of the sequence decremented by 1. We say that a walk *traverses* each vertex appearing on it, and it also *traverses* each edge between two consecutive vertices appearing on it. The first and the last vertex are called the *endpoints* of a walk. A walk is called *closed* if its endpoints are equal. The notions of (closed) walks can be naturally generalized to digraphs by requesting that for every two consecutive vertices of the sequence there is an arc from the first one to the second. We then say that the first vertex of a walk is its *beginning*, while the last is its *end*.

A *path* (or a *simple path*) in G is a subgraph of G formed by vertices and edges traversed by a walk in G that does not visit any vertex more than once. The first and the last vertex are called the *endpoints* of a path, and all the other vertices are *internal*. A *cycle* (or a *simple cycle*) is a subgraph of G formed by vertices and edges traversed by a closed walk in G of length at least 3 that does not visit any vertex more than once, apart from beginning and ending in the same vertex that is visited exactly twice. The *length* of a path or a cycle is the cardinality of its edge set; note that thus the length of a path or a cycle is equal to the length of the corresponding (closed) walk.

Analogically, using directed (closed) walks we can define *directed paths* and *directed cycles*. In the case of digraphs, however, we allow cycles of length 2.

Two paths are *edge-disjoint* or *vertex-disjoint* if their edge or vertex sets are disjoint, respectively. Two paths are *internally vertex-disjoint* if no internal vertex of one path lies on the other. These notions can be naturally generalized to digraphs; in this case we talk about *arc-disjointness* instead of edge-disjointness.

An undirected graph G is *acyclic*, or a *forest*, if it does not contain any cycle. It is moreover a *tree* if it is connected. Vertices of degree 1 in a forest are called *leaves*, while all the other vertices are called *internal*. A directed graph G is *acyclic*, or a *DAG* (for *directed acyclic graph*), if it does not contain any directed cycle. Equivalently, G admits a *topological ordering*, i.e., an ordering σ of $V(G)$ such that whenever $(v,w) \in E(G)$, then $v <_\sigma w$. Recall that given an acyclic digraph G, its topological ordering can be found in linear time.

An undirected graph G is *bipartite* if its vertex set can be partitioned into two sets X, Y (called *partite sets*) such that every edge of the graph has one endpoint in X and second in Y. Equivalently, a graph is bipartite if it does not contain any cycle of odd length. We often think of bipartite graphs as triples (X, Y, E), where X is the *left* partite set, Y is the *right* partite set, and E is the edge set.

Connectivity

A graph is *connected* if every two vertices can be connected via a path. Every graph can be in linear time decomposed into its *connected component*, i.e., inclusion-maximal connected subgraphs. If G is an undirected graph and $X \subseteq V(G)$, then we say that X is connected if $G[X]$ is connected.

A digraph is *weakly connected* if its underlying undirected multigraph is connected. For a directed graph G and two vertices $v, w \in V(G)$, we say that v and w *communicate* if there exist directed paths from v to w and from w to v. Note that communication is an equivalence relation. A digraph is *strongly connected* if every two vertices communicate. Equivalence classes of the relation of communication are called *strongly connected components*. For a digraph G, we have a natural structure of a DAG on the set of strongly connected components of G: there is an arc from component K to component K' if and only if $(v, v') \in E(G)$ for some $v \in K$ and $v' \in K'$. We call this DAG the *DAG of strongly connected components of* G. Recall that given a digraph G, its DAG of strongly connected components can be found in linear time [76].

Separations and cuts

In an undirected graph G, a *separation* is a pair of vertex subsets (A, B) such that $A \cup B = V(G)$ and $E(A \setminus B, B \setminus A) = \emptyset$. The *separator* of (A, B) is $A \cap B$, and the *order* of (A, B) is equal to $|A \cap B|$. For vertices $v \in A \setminus B$ and $w \in B \setminus A$, we say that separation (A, B) *separates* v and w. This corresponds to the observation that every path between v and w in G must contain a vertex from separator $A \cap B$. Thus, we may also say that the separator separates v and w. Observe that in this formalism, the classical Menger's theorem states that for every two different vertices v, w such that $vw \notin E(G)$, one can find either $k + 1$ internally vertex-disjoint paths between v and w, or a separation of order at most k that separates v and w.

We naturally extend the notation for separations to the directed setting as follows. Again, a separation is a pair of vertex subsets (A, B) such that $A \cup B = V(G)$ and $E(A \setminus B, B \setminus A) = \emptyset$. We define the separator and the order of a separation in the same manner as in the undirected setting. For $v \in A \setminus B$ and $w \in B \setminus A$, we say that (A, B) (or $A \cap B$) *separates* w *from* v. Again, in this notation the classical Menger's theorem states that for every two different vertices v, w such that $(v, w) \notin E(G)$, one can find either $k + 1$ internally vertex-disjoint directed paths from v to w, or a separation of order at most k that separates w from v.

While separations correspond to vertex-connectivity, for edge connectivity we use the terminology of *cuts*. For an undirected graph G, a *cut* in G is a partition (A, B) of $V(G)$. The *cutset* of (A, B) is $E(A, B)$, and the *order* of (A, B) is $|E(A, B)|$. Every cut of order at most k will be also called a *k-cut*. Again, we say that a cut or a cutset *separates* any vertex from A and any vertex from B. These notions are extended naturally to the directed setting, similarly as with separations. Note, however, that here the order of a cut (A, B) is the cardinality of its cutset $E(A, B)$, while we put no constraints on the size of $E(B, A)$. In this formalism the classical Menger's theorem for undirected graphs states that for an undirected graph G and any two different vertices $v, w \in V(G)$, one can either find $k + 1$ edge-disjoint paths between v and w, or a cut of order at most k that separates v and w. Similarly, for a directed graph G and any two vertices $v, w \in V(G)$, one can either find $k+1$ arc-disjoint paths from v to w, or a cut of order at most k that separates w from v

Tournaments and semi-complete digraphs

A simple digraph T is a *tournament* if for every pair of different vertices $v, w \in V(T)$, we have that *exactly* one of arcs (v, w) or (w, v) is present in $E(T)$. Similarly, T is *semi-complete* if for every pair of different vertices $v, w \in V(T)$, we have that *at least* one of arcs (v, w) or (w, v) is present in $E(T)$. A semi-complete digraph is a *transitive tournament* if it is acyclic. Equivalently, there exists an ordering σ of vertices of T such that $(v, w) \in E(T)$ if and only if $v <_\sigma w$. Note that the

DAG of strongly connected components of any semi-complete digraph is a transitive tournament.

2.2 Algorithmic frameworks

2.2.1 Parameterized complexity

A parameterized problem L is a subset of $\Gamma^* \times \mathbb{N}$ for some finite alphabet Γ. An instance of a parameterized problem consists of (x, k), where k is called the *parameter*. By $\tilde{L}$ we denote the *unparameterized version of* L, that is, a classical problem over alphabet $\Gamma \cup \{\mathbf{1}\}$ for some $\mathbf{1} \notin \Gamma$ where $y \in \tilde{L}$ if and only if y is of form $x\mathbf{1}^k$ for some $(x, k) \in L$.

We say that a problem L is *in class XP* if there exists an algorithm $\mathbb{A}$ that for a given instance (x, k) resolves whether $(x, k) \in L$ in time $f(k) \cdot |x|^{f(k)}$, where f is an arbitrary computable function of k. We moreover say that L is *fixed-parameter tractable* (FPT) if there exists an algorithm $\mathbb{B}$ that for a given instance (x, k) resolves whether $(x, k) \in L$ in time $f(k) \cdot p(|x|)$, where f is an arbitrary computable function of k and p is a polynomial of the input size. We also say that an algorithm or a running time are *XP* or *fixed-parameter tractable* if they satisfy respective conditions in the definitions above.

A parameterized problem L is *in non-uniform class XP* if there exists a sequence of algorithms $(\mathbb{A}_k)_{k\in\mathbb{N}}$ such that algorithm $\mathbb{A}_k$, given input x, resolves whether $(x, k) \in L$ in time $c_k \cdot |x|^{c_k}$, where c_k is some constant depending on k only. Similarly, L is *non-uniformly fixed-parameter tractable* if there exists a sequence of algorithms $(\mathbb{B}_k)_{k\in\mathbb{N}}$ such that algorithm $\mathbb{B}_k$, given input x, resolves whether $(x, k) \in L$ in time $c_k \cdot p(|x|)$, where c_k is some constant depending on k only and p is a polynomial of the input size.

A *parameterized reduction* from a parameterized problem L to a parameterized problem L' is an algorithm that, given an instance (x, k) of L, in time $f(k) \cdot |x|^{\mathcal{O}(1)}$ outputs an equivalent instance (x', k') of L' such that $k' \leq g(k)$, where f and g are some computable functions. Note that pipelining a parameterized reduction from L to L' with an FPT algorithm for L' yields an FPT algorithm for L.

The definitions of classes XP and FPT and of parameterized reductions can be naturally generalized to allow several parameters by allowing k to be a vector of nonnegative integers of some fixed length d. For instance, then the domain of functions f in the definitions is $\mathbb{N}^d$ rather than $\mathbb{N}$. Alternatively, the reader may think of a parameterization by multiple parameters as a parameterization by the sum of them.

Class SUBEPT, defined by Flum and Grohe [132, Chapter 16], consists of these fixed-parameter tractable problems which admit an algorithm working in time $f(k) \cdot p(|x|)$ on inputs (x, k), where $f \in 2^{o(k)}$ is a computable function and p is any polynomial. We also call such algorithms *subexponential parameterized algorithms*. Algorithms with running time $\mathcal{O}^*(\mathbf{tower}(q, p(k)))$ for any polynomial p will be called *q-times exponential*, and in particular algorithms with running time $\mathcal{O}^*(2^{p(k)})$ for any polynomial p will be called *single-exponential*. To avoid misunderstandings, algorithms with running time $\mathcal{O}^*(2^{\mathcal{O}(k)})$ will be called *simple single-exponential*.

We refer to the books of Downey and Fellows [119] and of Flum and Grohe [132] for further reading on parameterized complexity.

2.2.2 Kernelization

A *kernelization algorithm* for a parameterized problem $L \subseteq \Gamma^* \times \mathbb{N}$ is an algorithm that given $(x, k) \in \Gamma^* \times \mathbb{N}$ outputs in time polynomial in $|x| + k$ a pair $(x', k') \in \Gamma^* \times \mathbb{N}$ such that (i) $(x, k) \in L$

if and only if $(x', k') \in L$, and (ii) $|x'|, k' \leq g(k)$ for some computable function g. The output instance (x', k') is called the *kernel*, while function g is called the *size* of the kernel. We say that a problem *admits a polynomial kernel* if it admits a kernelization algorithm where the size of the kernel is a polynomial function of the parameter. The following lemma expresses the folklore observation stated and proved in Section 1.4.1.

Lemma 1 (Folklore, [119]). *Let L be a parameterized problem and assume that $\tilde{L}$ is decidable. Then L is fixed-parameter tractable if and only if it admits some kernelization algorithm.*

2.2.3 ETH and SETH

A propositional formula φ over a set of boolean variables X is in *conjunctive normal form* (CNF) if φ is a conjunction of m clauses, each being a disjunction of a number of literals, i.e., variables appearing in negated or non-negated form. The CNF-SAT problem is defined as follows: given a propositional formula in CNF on n variables and m clauses, determine whether there exists an evaluation of the variables into values true or false that satisfies φ. By restricting the number of literals that can appear in a single clause to q we arrive at the definition of the q-CNF-SAT problem.

For $q \geq 3$, let δ_q be the infimum of the set of constants c for which there exists an algorithm solving q-CNF-SAT in time $\mathcal{O}^*(2^{cn})$. In this definition we allow randomized two-sided error algorithms. ETH and SETH are then defined as follows.

Conjecture 2 (Exponential-Time Hypothesis, [195]). *The following holds: $\delta_3 > 0$.*

Conjecture 3 (Strong Exponential-Time Hypothesis, [195]). *The following holds: $\lim_{q\to\infty} \delta_q = 1$.*

We now recall the Sparsification Lemma of Impagliazzo et al. [196] and its immediate consequence.

Theorem 4 (Sparsification Lemma, Corollary 1 of [196]). *For all $\varepsilon > 0$ and positive q, there is a constant C so that any q-CNF formula φ with n variables can be expressed as $\varphi = \bigvee_{i=1}^{t} \psi_i$, where $t \leq 2^{\varepsilon n}$ and each ψ_i is a q-CNF formula with at most Cn clauses. Moreover, this disjunction can be computed by an algorithm running in time $2^{\varepsilon n} \cdot n^{\mathcal{O}(1)}$.*

Corollary 5 ([196]). *Unless ETH fails, there exists a constant $c > 0$ such that no algorithm for 3-CNF-SAT can achieve running time $\mathcal{O}^*(2^{c(n+m)})$. In particular, 3-CNF-SAT cannot be solved in $\mathcal{O}^*(2^{o(n+m)})$ time.*

2.2.4 Kernelization lower bounds

We now recall the cross-composition framework introduced by Bodlaender et al. [50] which builds upon Bodlaender et al. [45] and Fortnow and Santhanam [150]. We will not use this framework in this thesis; we provide the following formal details for reader's convenience, since we refer to lower bounds obtained via compositions in Section 1.4.2 and in Chapter 9.

Definition 6 (Polynomial equivalence relation, [50]). *An equivalence relation $\mathcal{R}$ on Σ^* is called a* polynomial equivalence relation *if (1) there is an algorithm that given two strings $x, y \in \Sigma^*$ decides whether $\mathcal{R}(x, y)$ in $(|x| + |y|)^{\mathcal{O}(1)}$ time; (2) for any finite set $S \subseteq \Sigma^*$ the equivalence relation $\mathcal{R}$ partitions the elements of S into at most $(\max_{x\in S} |x|)^{\mathcal{O}(1)}$ classes.*

Definition 7 (Cross-composition, [50]). *Let $L \subseteq \Sigma^*$ and let $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. We say that L* cross-composes *into Q if there is a polynomial equivalence relation $\mathcal{R}$ and an algorithm that, given t strings $x_1, x_2, \ldots x_t$ belonging to the same equivalence class of $\mathcal{R}$, takes time polynomial in $\sum_{i=1}^{t} |x_i|$ and computes an instance $(x^*, k^*) \in \Sigma^* \times \mathbb{N}$ such that (1) $(x^*, k^*) \in Q$ if and only if $x_i \in L$ for some $1 \leq i \leq t$; (2) k^* is bounded polynomially in $\max_{i=1}^{t} |x_i| + \log t$.*

Theorem 8 (Theorem 9 of [50]). *If $L \subseteq \Sigma^*$ is NP-hard under Karp reductions, and L cross-composes into a parameterized problem Q that has a polynomial kernel, then NP $\subseteq$ coNP/poly.*

The following result of Dell and van Melkebeek [102] is a sharpened version of the information-theoretical theorem of Fortnow and Santhanam [150] that led to development of the composition framework. We will find it useful in Chapter 9.

Theorem 9 ([102]). *Let $q \geq 3$ be an integer, ε be a positive real and $L \subseteq \{0,1\}^*$ be any language. If NP $\not\subseteq$ coNP/poly, then there is no polynomial-time algorithm that takes a q-CNF formula φ on n variables and outputs a binary string u of length $\mathcal{O}(n^{q-\varepsilon})$ such that $u \in L$ if and only if φ is satisfiable.*

Such algorithms as the one refuted in Theorem 9 are called *compression algorithms* in order to distinguish them from kernelization algorithms, where we additionally require that the target language is the same as the source language. We remark that the original statement of Dell and van Melkebeek is more general and refutes even existence of protocols certifying satisfiability with total communication cost $\mathcal{O}(n^{q-\varepsilon})$; we refer to [102] for details. Theorem 9 was used by Dell and van Melkebeek to provide tight bounds on kernel sizes for some classical parameterized problems, and is the crucial ingredient of the later framework of *weak compositions* of Dell and Marx [101] and of Hermelin and Wu [188].

2.3 Width measures and the topological theory for graphs

2.3.1 Containment relations

Let H and G be undirected graphs. We say that H is a *minor* of G if there exists a family $(B_v)_{v \in V(H)}$ of disjoint subsets of $V(G)$ such that (i) B_v is connected for each $v \in V(H)$, and (ii) if $vw \in E(H)$, then $v'w' \in E(G)$ for some $v' \in B_v$ and $w' \in B_w$. The sets B_v are called *branch sets* and the family $(B_v)_{v \in V(H)}$ is called the *minor model* of H in G.

We say that H is a *topological subgraph* of G, or is *topologically contained (embedded)* in G, if there exists a mapping η such that (i) for $v \in V(H)$, $\eta(v)$ are distinct vertices of G, (ii) for $vw \in E(H)$, $\eta(vw)$ is a path between $\eta(v)$ and $\eta(w)$, and (iii) all the paths $\eta(vw)$ for $vw \in E(H)$ are pairwise internally vertex-disjoint. If we relax the requirement of vertex-disjointness to edge-disjointness, we arrive at the definition of *immersion*.

It is easy to observe that the minor, topological subgraph and immersion relations are reflexive, transitive, and antisymmetric, and hence they constitute partial orders of the set of undirected graphs.

For a graph H, a *subdivision* of H is any graph that can be obtained from H by repeated use of the following operation: take any edge $vw \in E(H)$, and substitute it with edges vu and uw for a newly introduced vertex u. Note that G contains H as a topological subgraph if and only if G contains a subdivision of H as a subgraph.

For a graph G and an edge $vw \in E(G)$, we define the *edge contraction* operation as follows. Substitute vertices v and w with a newly introduced vertex u, and put ux into the edge set for exactly these vertices $x \in V(G) \setminus \{v, w\}$ for which $vx \in E(G)$ or $wx \in E(G)$. It can be easily observed that G contains H as a minor if and only if H can be obtained from G by repeated use of vertex deletions, edge deletions, and edge contractions. If we additionally require that any contracted edge has at least one endpoint of degree 2, then we obtain the notion of a topological subgraph. Immersion can be also defined in terms of graph operations using yet another operation called *lift*; we refer to [27, 166] for details.

The notions of a topological subgraph and immersion can be naturally lifted to the directed setting by replacing paths with directed paths. There is no clear consensus about generalization of the notion of a minor. In Part II of this thesis we use one of the propositions that has been presented recently by Kim and Seymour [216]. We refer to Section 5.2.2 for a unified formalization of containment relations in directed graphs.

2.3.2 Treewidth and pathwidth

A *tree decomposition* of an undirected graph G is a tree $\mathcal{T}$ in which each vertex $x \in V(\mathcal{T})$ has an assigned set of vertices $B_x \subseteq V(G)$ (called a *bag*) such that the following properties are satisfied:

- $\bigcup_{x \in V(\mathcal{T})} B_x = V(G)$;
- for each $uv \in E(G)$, there exists an $x \in V(\mathcal{T})$ such that $u, v \in B_x$;
- for each $v \in V(G)$, the set of vertices of $\mathcal{T}$ whose bags contain v induces a connected subtree of $\mathcal{T}$.

The *width* of a tree decomposition $\mathcal{T}$ is $\max_{x \in V(\mathcal{T})} |B_x| - 1$, and the treewidth of a graph G, denoted $\mathbf{tw}(G)$, is the minimum treewidth over all possible tree decompositions of G. To distinguish vertices of $\mathcal{T}$ from vertices of G we often call the vertices of $\mathcal{T}$ *nodes*. By restricting $\mathcal{T}$ to be a path we arrive at the notions of a *path decomposition* and *pathwidth*, denoted $\mathbf{pw}(G)$.

2.3.3 Cliquewidth

A *labeled graph* is a pair (G, α), where G is a graph and $\alpha : V(G) \to \{1, 2, \ldots, k\}$ is a labeling function that associates with each vertex of G one of k different labels. We define three operations on labeled graphs:

- **Disjoint union** $\oplus$ is defined as

$$(G_1, \alpha_1) \oplus (G_2, \alpha_2) = (G_1 \cup G_2, \alpha_1 \cup \alpha_2).$$

  In other words, we take the disjoint union of graphs G_1 and G_2, and define the labeling as the union of original labelings.

- **Join** $\eta_{i,j}(\cdot)$ for $i, j \in \{1, 2, \ldots, k\}$, $i \neq j$, is defined as

$$\eta_{i,j}((G, \alpha)) = (G', \alpha),$$

  where G' is G after introducing all possible edges having one endpoint labeled with i and second with j.

- **Relabel** $\rho_{i\to j}(\cdot)$ for $i, j \in \{1, 2, \ldots, k\}$, $i \neq j$, is defined as

$$\rho_{i\to j}((G, \alpha)) = (G, \alpha'),$$

  where α' is α with all the values i substituted with j.

A *clique expression* is a term that uses operators $\oplus$, $\eta_{i,j}(\cdot)$, $\rho_{i\to j}(\cdot)$, and constants $\iota_1, \iota_2, \ldots, \iota_k$ that represent one-vertex graphs with the only vertex labeled with $1, 2, \ldots, k$, respectively. In this manner a clique expression *constructs* some labeled graph (G, α). The cliquewidth of a graph G (denoted $\mathbf{cw}(G)$) is the minimum number of labels needed in a clique expression that constructs G (with any labeling).

2.3.4 Branch decompositions, branchwidth, rankwidth, and carving-width

Many definitions of width measures of graphs follow formalism introduced by Robertson and Seymour for branchwidth and branch decompositions [284]. Let X be any set and μ be any function that maps partitions of X into two subsets to nonnegative reals. We will call μ the *cut function* and require it to be symmetric, i.e., $\mu((A, B)) = \mu((B, A))$ for any partition (A, B) of X. A *branch decomposition* is a tree $\mathcal{T}$ with elements of X associated with leaves of $\mathcal{T}$, and every internal vertex of degree exactly 3. For any edge $e \in E(\mathcal{T})$, removing e from $\mathcal{T}$ breaks $\mathcal{T}$ into two connected components; let (A_e, B_e) be the partition of elements of X with respect to the component to which they fall. Then the *width* of decomposition $\mathcal{T}$ is defined as $\max_{e\in E(\mathcal{T})} \mu((A_e, B_e))$. The *$\mu$-width* of X is defined as the minimum possible width of a branch decomposition of X.

By choosing different sets X and cut functions μ we obtain different width notions of combinatorial objects. For instance, if for an undirected graph G we take $X = E(G)$ and define $\mu((A, B))$ to be the number of vertices of G that are simultaneously adjacent to an edge from A and to an edge from B, we arrive at the definition of branchwidth [284] (denoted $\mathbf{bw}(G)$). On the other hand, if for an undirected graph G we take $X = V(G)$ and simply put $\mu((A, B)) = |E(A, B)|$, we arrive at the definition of *carving-width* [298]. Finally, taking $X = V(G)$ and $\mu((A, B))$ to be the rank over $\mathbb{F}_2$ of the binary matrix of adjacency between A and B leads to the definition of *rankwidth* [265] (denoted $\mathbf{rw}(G)$).

2.3.5 Linear width measures and cutwidth

Let X be a set of size n, and μ be a cut function as in Section 2.3.4; this time, we do not require μ to be symmetric. For an ordering $\sigma = (x_1, x_2, \ldots, x_n)$ of X, we define the *width* of σ as

$$\max_{i=0,1,\ldots,n} \mu(\{x_1, x_2, \ldots, x_i\}, \{x_{i+1}, x_{i+2}, \ldots, x_n\}).$$

The *linear μ-width* of X is defined as the minimum possible width among orderings of X. The *cutwidth* of a graph G (denoted $\mathbf{ctw}(G)$) is defined by taking $X = V(G)$ and $\mu((A, B)) = |E(A, B)|$.

2.3.6 Models of logic on graphs

FO is First-Order logic on graphs. The syntax of **FO** consists of

- logical connectives $\vee$, $\wedge$, $\neg$, $\Leftrightarrow$, $\Rightarrow$, with standard semantics;
- variables for vertices;

- quantifiers $\forall$, $\exists$ that can be applied to these variables;
- and the following two binary relations:
  1. $\mathbf{E}(u, v)$, where u, v are vertex variables, and the semantics is that $\mathbf{E}(u, v)$ is true if and only if edge uv is present in the graph;
  2. equality of variables.

$\mathbf{MSO}_1$ is Monadic Second-Order logic on graphs with quantification over subsets of vertices but not of edges. The syntax of $\mathbf{MSO}_1$ extends that of **FO** by the following features:

- there are variables for vertices and for sets of vertices;
- quantifiers $\forall$, $\exists$ as well as equalities can be applied to all these variables;
- we add binary relation $u \in U$, where u is a vertex variable and U is a vertex set variable, with standard semantics.

$\mathbf{MSO}_2$ is Monadic Second-Order logic on graphs with quantification both over subsets of vertices and of edges. The syntax of $\mathbf{MSO}_2$ consists of

- logical connectives $\vee$, $\wedge$, $\neg$, $\Leftrightarrow$, $\Rightarrow$, with standard semantics;
- variables for vertices, edges, subsets of vertices, and subsets of edges;
- quantifiers $\forall$, $\exists$ that can be applied to these variables;
- and the following three binary relations:
  1. $u \in U$, where u is a vertex (edge) variable and U is a vertex (edge) set variable, with standard semantics;
  2. $\mathbf{inc}(u, e)$, where u is a vertex variable and e is an edge variable, and the semantics is that $\mathbf{inc}(u, e)$ is true if and only if edge e is incident with vertex u;
  3. equality of variables.

Thus, formally $\mathbf{MSO}_2$ does not extend $\mathbf{MSO}_1$, but it is easy to see that every formula of $\mathbf{MSO}_1$ can be translated to an equivalent formula of $\mathbf{MSO}_2$. The set variables are also called *monadic* (vertex/edge) variables.

FO, $\mathbf{MSO}_1$, and $\mathbf{MSO}_2$ can be naturally generalized to digraphs by

- replacing edge (set) variables with arc (set) variables;
- for **FO** and $\mathbf{MSO}_1$, replacing relation $\mathbf{E}(u, v)$ with relation $\mathbf{A}(u, v)$, with semantics that $\mathbf{A}(u, v)$ is true if and only if arc (u, v) is present in the digraph;
- for $\mathbf{MSO}_2$, replacing relation $\mathbf{inc}(u, a)$ with relations $\mathbf{head}(u, a)$ and $\mathbf{tail}(u, a)$, with semantics that $\mathbf{head}(u, a)$ ($\mathbf{tail}(u, a)$) is true if and only if u is the head (tail) of the arc a.

Formally, formulas of **FO**, $\mathbf{MSO}_1$, $\mathbf{MSO}_2$ are evaluated on graphs seen as logical structures. In case of **FO** and $\mathbf{MSO}_1$ the domain is the vertex set of the graph, while the signature contains binary relation **E** with interpretation that $\mathbf{E}(u,v)$ holds if and only if there is an edge between u and v in the graph. In case of $\mathbf{MSO}_2$ the domain is the disjoint union of the vertex set and the edge set of the graph, while the signature contains unary relations $\mathbf{Vertex}(\cdot)$ and $\mathbf{Edge}(\cdot)$ testing whether a given element is a vertex or an edge (usually used implicitly), and binary relation $\mathbf{inc}(\cdot,\cdot)$ with interpretation that $\mathbf{inc}(u,e)$ holds if and only if $\mathbf{Vertex}(u)$, $\mathbf{Edge}(e)$, and vertex u is incident to edge e. We can also enrich the signature with constants representing single vertices (or edges, in case of $\mathbf{MSO}_2$), and unary relations representing prespecified sets of vertices (or edges, in case of $\mathbf{MSO}_2$). Assume that $\overline{X}$ is a vector of symbols enriching the signature, with interpretation $\overline{A}$ in a graph G. Then for a formula $\varphi(\overline{X})$ of $\mathbf{FO}, \mathbf{MSO}_1, \mathbf{MSO}_2$ with free variables from $\overline{X}$ we say that φ *is satisfied in* G *with prescribed interpretation* $\overline{A}$ if and only if $\langle G, \overline{A}\rangle \models \varphi(\overline{A})$. Investigating logic on graphs is not the main topic of this thesis. Therefore, we usually make statements about formulas on graphs via this notion of satisfaction, rather than using the formal language of logical structures. The translation to the language of logical structures is in each case obvious.

An example of a graph property that can be expressed in **FO** is existence of an independent set of size k, for any fixed k. Indeed, we quantify existentially k vertices of the graph and check that they are pairwise different and non-adjacent. Note, however, that the length of such a formula is $\mathcal{O}(k^2)$.

An example of a graph property expressible in $\mathbf{MSO}_1$ is 3-colorability. Indeed, we quantify existentially 3 monadic variables X, Y, Z corresponding to vertices of the three colors, and check (i) that they form a partition of the vertex set (each vertex belongs to exactly one of the sets), and (ii) that each of them induces an independent set (no two vertices from the same set are adjacent).

An example of a graph property expressible in $\mathbf{MSO}_2$ is hamiltonicity. Indeed, we quantify existentially one monadic edge variable C corresponding to the edge set of the hamiltonian cycle, and check (i) that C induces a connected graph (there is no partition of C into nonempty C_1 and C_2 such that no vertex is incident both to an edge of C_1 and to an edge of C_2), and (ii) that each vertex is incident to exactly two edges of C.

2.3.7 Well-quasi orderings

A *quasi-order* is a pair $(P, \leq)$, where $\leq$ is a reflexive and transitive relation on P. Note that partial orders are exactly quasi-orders where we additionally require that the relation is also anti-symmetric. A quasi-order $(P, \leq)$ is a *well-quasi-ordering* (WQO) if for any infinite sequence of elements $x_1, x_2, x_3, \ldots \in P$ there exist indices $i < j$ such that $x_i \leq x_j$. An *infinite decreasing sequence* is a sequence of elements $x_1 > x_2 > x_3 > \ldots$. A subset $S \subseteq P$ is *closed under* $\leq$ if whenever $x \leq y$ and $y \in S$, then also $x \in S$. We say that S is *characterized by obstruction set* $\mathcal{F}_S$ if $S = \{x \in P \mid \neg\exists_{f \in \mathcal{F}_S}\ f \leq x\}$. The following folklore observation has been already expressed in Section 1.4.1.

Lemma 10 (Folklore)**.** *Let $(P, \leq)$ be a quasi-order. Then $(P, \leq)$ is a well-quasi-ordering if and only if P does not contain any infinite decreasing sequences and every subset of P closed under $\leq$ is characterized by a finite obstruction set.*

Chapter 3

Width parameters of graphs

3.1 The Graph Minors project

We first survey the achievements of the Graph Minors project of Robertson and Seymour, which is the origin of the interest in width measures of graphs. In Part II we consider a similar, but much simpler theory for semi-complete digraphs. We hope that the following section will help the reader see how the general concepts and results of the Graph Minors project were an inspiration for our approach.

Recall that the main achievement of the work of Robertson and Seymour is the Graph Minors Theorem:

Theorem 11 (Graph Minors Theorem, [287])**.** *The minor order is a well-quasi-ordering of the class of undirected graphs.*

For a definition of a minor, see Section 2.3.1. We begin with an extremely quick overview of the proof of Theorem 11; an interested reader is also invited to the book of Diestel [112] for a more detailed description. Our aim is to introduce the most important concepts and explaining their origin in order to prepare the ground for further discussions. Then we take a closer look at three topics that are of our main interest: the Excluded Grid Minor Theorem, the Decomposition Theorem for H-minor free graphs, and the algorithmic aspects of the project. Finally, we survey what is known about the directed setting.

3.1.1 An express overview of the proof

The high-level strategy of the proof of Robertson and Seymour is to prove the theorem for larger and larger subclasses of undirected graphs. Consider first the theorem restricted to planar graphs. Take any sequence of planar graphs $G_0, G_1, G_2, \ldots$ and for the sake of contradiction assume that there are no indices $i < j$ such that $G_i \preceq G_j$, where $\preceq$ denotes the minor order. In particular all the graphs G_j for $j \geq 1$ do not contain $H := G_0$ as a minor; we say that they are *H-minor free*. Therefore, we need to answer the following natural question: how do graphs excluding some fixed-size planar graph H look like?

The answer to this problem comes via the so-called *Excluded Grid Minor Theorem*. In the following, by a *$k \times k$ grid* we mean a graph with vertex set $\{1, 2, \ldots, k\} \times \{1, 2, \ldots, k\}$ where (i, j) and (i', j') are neighbors if and only if $|i - i'| + |j - j'| = 1$.

Theorem 12 (Excluded Grid Minor Theorem, [281], see also [112, 113, 290])**.** *There is a function $r(k)$ such that every graph with treewidth more than $r(k)$ contains a $k \times k$ grid as a minor.*

It can be easily observed that every planar graph H is a minor of a $f(|H|) \times f(|H|)$-grid for some function f. Hence, by excluding a *constant size* planar graph H as a minor, we bound the treewidth of a graph by a *constant*. Thus, all the graphs $G_1, G_2, G_3, \ldots$ have in fact constant treewidth, and we have reduced the problem from planar graphs to planar graphs of constant treewidth. However, the theorem for trees have been already proven by Kruskal [233] (see also a shorter proof of Nash-Williams [256]), and the proof can be lifted to graphs of constant treewidth with some technical efforts [282].

By mimicking this strategy, in order to prove the general statement we need to examine the structure of H-minor free graphs for general H. Note that planar graphs are exactly $\{K_{3,3}, K_5\}$-minor free graphs, so there is hope that for constant-size H, H-minor free graphs will be some sort of generalization of planar graphs, to which the argumentation from the planar case can be lifted. And indeed, Robertson and Seymour have proven the so-called *Decomposition Theorem* [286] that intuitively says the following: a graph excluding a fixed H as a minor can be obtained from graphs that can be (almost) embedded without crossings on a surface of constant genus, by gluing them along small interfaces. The formal definition is somewhat technical and is not relevant for the content of this thesis; we discuss its intuition in Section 3.1.3.

Thus, the 'only' thing we need to do is to lift the proof from the planar case to the case of graphs of bounded genus, then to handle almost embeddability, and finally lift everything to H-minor free graphs via the Decomposition Theorem. The first lift, presented in [283], is probably the most difficult one. It requires a careful induction on genus and several auxiliary parameters, and application of the following WIN/WIN approach. Very roughly speaking, we try to find a short cycle in the graph such that cutting the surface along this cycle reduces genus. If such a cycle exists, we may apply induction hypothesis. Otherwise, the graph under consideration is complicated enough to admit every constant-size graph of the same genus as a minor. The latter two lifts are more of technical nature, but of course they require a number of non-trivial observations. Summarizing, the proof consists of the following sequence of lifts:

$$\begin{array}{ccccccccc} \textbf{Trees} & \rightarrow & \textbf{Bounded treewidth} & \rightarrow & \textbf{Planar} & \rightarrow & \textbf{Bounded genus} & \rightarrow \\ & & \textbf{Almost embeddable} & \rightarrow & \textbf{H-minor free} & \rightarrow & \textbf{General} & \end{array}$$

The most important, and also the longest part of the proof is the Decomposition Theorem itself. Intuitively, the crucial step of the reasoning is to understand how non-existence of a purely combinatorial object, such as a minor model, can be used to construct an embedding of a graph on a surface. Finally, we remark that the ideas introduced in the proof of the Graph Minors theorem can used to show that the immersion order is also a well-quasi-ordering of the class of undirected graphs [288].

3.1.2 The Excluded Grid Minor Theorem

Because of its importance in the Graph Minors project and many applications outside it, the Excluded Grid Minor Theorem, i.e. Theorem 12, was studied intensively. Theorem 12 precisely identifies the sole reason of not admitting a tree decomposition of small width — this reason is containing a large grid-like structure that cannot be cut in halves by a small separator. Because of numerous algorithmic applications that we will review in Section 3.2.4, pursuit of as good estimates on function r as possible is a fundamental problem.

The first proof given by Robertson and Seymour provided only a non-elementary bound on function r [281]. Later, Robertson, Seymour, and Thomas gave an alternative proof of the Excluded Grid Minor Theorem from which a bound $r(k) \leq 2^{\mathcal{O}(k^5)}$ follows (see also much simpler proofs achieving slightly weaker bounds [112, 113]), and conjectured that the actual relation should be polynomial [290]. However, the best known lower bound for $r(k)$ is $\Omega(k^2 \log k)$ [290]. Further improvements have been given by Kawarabayashi and Kobayashi [207] ($r(k) \leq 2^{\mathcal{O}(k^2 \log k)}$) and by Leaf and Seymour [237] ($r(k) \leq 2^{\mathcal{O}(k \log k)}$). Very recently, Chekuri and Chuzhoy have claimed the first polynomial bound of $r(k) \leq \mathcal{O}(k^{228})$ [63], thus resolving the problem of Robertson, Seymour, and Thomas.

As we will see in Section 3.2, from the point of view of applications it is important to provide stronger versions of the Excluded Grid Minor Theorem for classes of graphs with topological constraints, like planar graphs, graphs of bounded genus and H-minor free graphs. Already Robertson, Seymour, and Thomas [290] observed that if we assume that the graph is planar, then in Theorem 12 one can use $r(k) = 6k-5$. Demaine and Hajiaghayi [107] have extended this result to H-minor free graphs: for every fixed H one can take $r(k) = c_H \cdot k$ for some constant c_H depending on H only. Recently, Gu and Tamaki [179] have shown that in the planar case one can substitute $6k-5$ with $\frac{9}{2}k$.

3.1.3 The Decomposition Theorem

Let us recall that the Decomposition Theorem states that a graph excluding a fixed graph H as a minor can be constructed by iteratively gluing graphs that are (almost) embeddable on a surface of constant genus along small interfaces. First, we need to define what does it mean to be almost embeddable. We remark here that different notions of almost embeddability can be found in the literature, thus the corresponding versions of the Decomposition Theorem differ in details.

Slightly informally speaking, we say that a graph G is (p, q, r, s)-*almost embeddable* if there exists a set of vertices Z (called *apices*), $|Z| \leq s$, such that $G \setminus Z$ can be embedded on a surface of genus r apart from at most q *vortices*. Each vortex R is a subgraph of $G \setminus Z$ of pathwidth at most p that is to be embedded into a disk on the surface in such a manner that the disk can contain crossings of edges of R, but the path decomposition of R must be 'glued' along disk's boundary. In other words, an almost embeddable graph is a graph that can be embedded into a surface of constant genus apart from (i) a constant number of special vertices that cannot be fit into the embedding, and (ii) a constant number of singularities on the surface that can be, however, controlled by path decompositions of constant width. For precise and formal statements of different versions of almost embeddability see e.g. [175, 212, 286].

Finally, we need to define the notion of gluing along small interfaces. Probably the cleanest way is to formalize this concept via tree decompositions. For a tree decomposition $\mathcal{T}$ of G and a node $x \in V(\mathcal{T})$, by *torso of* x, denoted τ_x, we mean graph $G[B_x] \cup \bigcup_{xy \in E(\mathcal{T})} K[B_x \cap B_y]$, where $K[A]$ stands for a clique on vertices from A, for $A \subseteq V(G)$. In other words, we take the graph induced by the bag at x and for every neighboring node y we put a complete graph on the vertices shared by B_x and B_y (on the so-called *adhesion*). Finally, we can state the Decomposition Theorem of Robertson and Seymour formally.

Theorem 13 (H-minor-free Decomposition Theorem, [286])**.** *For every graph H there are constants p, q, r, s such that any graph G on n vertices admits a tree decomposition $\mathcal{T}$ where every torso is (p, q, r, s)-almost embeddable. Moreover, given G and H, this decomposition along with the corresponding embeddings can be computed in $f(|H|) \cdot n^3$ time, where $n = |V(G)|$ and f is some*

computable function.[1]

Existence of an almost embedding is an obstruction for admitting large clique minors, cf. [199]. In particular, since τ_x have to be (p, q, r, s)-almost embeddable for constants p, q, r, s depending on H only, then the cliques introduced by the torso operation must be also of constant size. Therefore, decomposition $\mathcal{T}$ given by Theorem 13 has to have constant-size adhesions, where the constant depends on H only.

Let us remark that Robertson and Seymour did not give any bound on the obtained functions p, q, r, s, f, but even very rough estimates show that they are astronomical, and in particular non-elementary. Recently, Kawarabayashi and Wollan [212] gave a simpler proof of the Decomposition Theorem from which more explicit bounds on p, q, r, s can be derived.

The Decomposition Theorem is a very convenient tool for lifting results from graphs embeddable on a surface of constant genus to graphs excluding a constant-size H as a minor. One needs to handle three additional technical aspects: apices, vortices, and gluing graphs along small adhesions. Since there are only a constant number of apices in each torso, and vortices are controllable via path decompositions of constant width, many algorithms for graphs of constant genus can be generalized to almost embeddable graphs with some technical efforts. To incorporate the top-most tree decomposition layer of the theorem, one usually uses the algorithm for almost embeddable graphs to construct a global dynamic programming routine working on $\mathcal{T}$. This general framework has been successfully applied to many problems in parameterized complexity and approximation, see for example [108, 109, 110, 131, 144]. It should be mentioned that a vast majority of the problems to which the framework can be applied is completely intractable in the general case. Therefore, via the Decomposition Theorem we can expand greatly the class of inputs on which we can handle these problems efficiently.

It is worth mentioning that Grohe and Marx [177] have recently obtained a version of the Decomposition Theorem for graphs excluding a fixed H as a topological subgraph; observe that this is a larger class of graphs than H-minor free. Informally, while all the torsos in the Decomposition Theorem of Robertson and Seymour are almost embeddable, Grohe and Marx allow also the second type of torsos, namely having *almost bounded degree*. That is, after removing a constant number of vertices the torso becomes a graph of constant maximum degree. The new decomposition theorem has been already used to lift some results from the H-minor free setting to the H-topological-subgraph-free setting [145].

Let us conclude by remarking that all the considered graph classes, including H-minor free graphs and H-topological-subgraph free graphs, are *sparse*. In particular, for every H there exists a constant $d = d(H)$ such that every H-minor and H-topological-subgraph free graph G is d-degenerate [112]; recall that it means that every subgraph of G contains a vertex of degree at most d. Since d-degeneracy of G implies that $|E(G)| \leq d|V(G)|$, as a corollary we obtain that in H-minor free and H-topological-subgraph free graphs the number of edges is linear in the number of vertices. However, some works indicate that it is not d-degeneracy, but rather constraints of topological nature that enable us to solve problems efficiently, since many results cannot be lifted to general d-degenerate graphs [87, 96].

[1] The algorithmic statement was not claimed by Robertson and Seymour, but it follows from the later work, e.g. [212].

3.1.4 Algorithmic problems

One of the main algorithmic corollaries of the Graph Minors project is the FPT algorithm for testing the minor relation.

Theorem 14 ([285]). *There exists an algorithm which, given graphs G, H, checks whether H is a minor of G in time $f(|H|) \cdot n^3$, where $n = |V(G)|$ and f is some computable function.*

Note that Theorem 14 is crucial in non-constructive proofs of existence of non-uniform FPT algorithms via the Graph Minors Theorem, which we discussed in Section 1.4.1. The idea of the proof of Theorem 14 is to find an algorithm for the related Vertex Disjoint Paths problem and use the same concepts for minor testing. Due to its importance in this thesis, we state the Vertex Disjoint Paths problem explicitely.

Vertex Disjoint Paths	
Input:	An undirected graph G and k pairs of pairwise distinct terminals $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$
Parameter:	k
Question:	Can one find vertex-disjoint paths $P_1, P_2, \ldots, P_k$ so that P_i connects s_i with t_i, for $i = 1, 2, \ldots, k$?

By relaxing the requirement of vertex-disjointness to edge-disjointness we obtain the Edge Disjoint Paths problem. Note that this problem is easily reducible to Vertex Disjoint Paths by taking the line graph. Both problems have been shown to be NP-hard by Karp [204].

Theorem 15 ([285]). *The* Vertex Disjoint Paths *and* Edge Disjoint Paths *problems can be solved in time $f(k) \cdot n^3$, where $n = |V(G)|$ and f is some computable function.*

We remark that recently Kawarabayashi et al. [208] have improved the polynomial factor of the algorithms of Theorems 14 and 15 to quadratic in terms of n.

The original proof of Theorem 15 of Robertson and Seymour is one of the first applications of a WIN/WIN approach connected to treewidth, and probably also one of the most involved ones. First, we try to find a large (in terms of k) clique minor in G. If we succeed, then it is not very difficult to identify a vertex in this minor that can be safely removed, since some hypothetical solution can be assumed not to traverse it. Otherwise, we try to find a tree decomposition of G of small width. Again, if we succeed, then we can employ a simple dynamic program on this tree decomposition.

The only situation left is when we know that G does not admit easily identifiable clique minors, but it has also large treewidth. If we try to apply a combination of the Excluded Grid Minor Theorem and the Decomposition Theorem to this case, then we see that G should be formed from large graphs embeddable on simple surfaces, which in particular contain large grid-like structures. However, the Decomposition Theorem has been developed only later in the series. For this reason, Robertson and Seymour have proven the so-called *Weak Structure Theorem* [285] that asserts that in this situation G must contain a large *flat wall*: a grid-like structure that can be (almost) embedded in the plane so that only the boundary of the wall is adjacent to the rest of the graph. Then, using an involved argument Robertson and Seymour show how to identify an irrelevant vertex in a flat wall [289]. After deleting this vertex we restart the whole algorithm.

FPT algorithms for Vertex Disjoint Paths and Edge Disjoint Paths immediately give XP algorithms for testing topological containment and immersion relations: we just need to guess

the images of the vertices of H and run the algorithm testing whether images of edges of H can be constructed. However, fixed-parameter tractability of these problems has been resolved only recently by Grohe et al. [174], after resisting attacks as a long-standing open problem. The line of reasoning of Grohe et al. follows in principles that of Robertson and Seymour.

Function f in the algorithm of Robertson and Seymour is again astronomical, and finding algorithms with sensible dependence on k is a major open problem. For the case of planar graphs, Adler et al. [2] have given an algorithm with double-exponential dependence on k, and provided examples suggesting that beating this running time would require a major change of the strategy.

3.1.5 The theory of graph minors for directed graphs

It is natural to ask to what extent the achievements of the Graph Minors project can be extended to directed graphs. Obviously, both the Vertex Disjoint Paths and Edge Disjoint Paths problems can be generalized to digraphs by requesting P_i to be a directed path from s_i to t_i, for $i = 1, 2, \ldots, k$. The same generalization can be also performed for the notions of topological containment and immersion. Since it is not clear how the notion of a minor can be naturally generalized to directed graphs, it is topological containment and immersion that are two main containment relations considered in the directed setting.

Unfortunately, the classical result of Fortune, Hopcroft and Wyllie [151] states that both Vertex Disjoint Paths and Edge Disjoint Paths are already NP-hard for $k = 2$, and thus topological containment testing and immersion testing are NP-hard for fixed-size digraphs H. Hence, one does not expect existence of an algorithmically interesting topological theory for general digraphs. Moreover, it is not clear how to define an analogue of treewidth in the directed setting. We review some of the propositions in Section 3.4, but none of them shares all the nice properties of treewidth. For these reasons, the research concentrated on designing algorithms for topological problems in subclasses of directed graphs. Essentially, three major examples of subclasses have been considered.

First, mirroring the approach of the Graph Minors project it is natural to ask whether topological restrictions help in solving the Vertex (Edge) Disjoint Paths problems. Of particular interest is the case of **planar digraphs**. And indeed, Schrijver has shown that in this case an XP algorithm can be designed for the Vertex Disjoint Paths problem [293]. Whether this result could be strengthened to fixed-parameter tractability had been a major open problem in the field for several years [119], until very recently Cygan et al. [90] have claimed an FPT algorithm with running time $\mathcal{O}^*(2^{2^{k^{\mathcal{O}(1)}}})$. The approach of Cygan et al. mimics the algorithm for the undirected case of Adler et al. [2], but introduces a new kind of decomposition that is better suited for the algebraic tools of Schrijver.

Second, if we require the digraph to be **acyclic**, then the Vertex (Edge) Disjoint Paths problems can be solved in XP time [151], thus yielding immediately XP algorithms for topological containment and immersion testing. Unfortunately, Slivkins [300] has shown that both problems are in fact W[1]-hard, thus completing the picture of their parameterized complexity on DAGs. The author is not aware of any significant structural results for general DAGs in the context of topological problems.

Third, a sound containment theory, resembling the Graph Minors project on a smaller scale, can be constructed for the class of **tournaments**, or more generally **semi-complete digraphs**. Part II of this thesis treats of our contribution to this theory; hence, we refer to the introductory sections of this part for a broader discussion.

3.2 Treewidth and its applications

In this section we survey results on treewidth, which is arguably the most important width measure of graphs. We hope that this will help the reader to find similarities between the results on treewidth contained in the literature and our study of width measures of semi-complete digraphs in Part II. We first discuss different algorithms for computing treewidth of a given graph. Then we review connections between treewidth and logic, explaining relations with $\mathbf{MSO}_2$ and applications in model checking **FO** on sparse graph classes. Finally, we survey applications of treewidth and the Excluded Grid Minor Theorem in parameterized complexity, with particular focus on bidimensionality.

3.2.1 Algorithms for treewidth

The most classical algorithm for computing treewidth has been given by Robertson and Seymour in the Graph Minors series [285]; see also a self-contained explanation in the textbook of Kleinberg and Tardos [218]. This algorithm can be today called an *FPT approximation*. That is, given a graph G on n vertices and an integer k, the algorithm works in $\mathcal{O}(3^{3k}\cdot n^2)$ time and either returns a decomposition of width at most $4k+3$, or an obstacle for admitting a decomposition of width at most k, called *k-unbreakable set*. This approximation framework, i.e., that the algorithm is allowed to compute a decomposition of width slightly larger than the given target, is commonly used when computing width measures. Relaxation of the requirement on the target width allows us to design faster algorithms, while usually obtaining a decomposition of width bounded by a low function of the optimal width is harmless for the algorithm's applications. We will use this framework often in Part II.

In the early stages of the study of treewidth as a graph parameter, the research concentrated on improving the polynomial factors of algorithms computing it. An algorithm with running time $2^{\mathcal{O}(k\log k)}\cdot n\log^2 n$ and outputting a decomposition of width at most $8k+7$ was proposed by Lagergren et al. [235], which was later trimmed to $2^{\mathcal{O}(k\log k)}\cdot n\log n$ by Reed [275]. We remark here that Reed does not give the approximation ratio of his algorithm explicitely, but it can be shown that the output decomposition has width at most $8k+\mathcal{O}(1)$. Finally, Bodlaender [42] gave the first linear-time algorithm for treewidth, working in $k^{\mathcal{O}(k^3)}\cdot n$ time. The algorithm of Bodlaender is *exact* in the sense that it either provides a tree decomposition of width at most k, or correctly claims that such a decomposition does not exist. The previously known best algorithm for computing treewidth exactly was due to Arnborg et al. [15] and worked in XP time.

As far as approximation in polynomial time is concerned, the currently best algorithm of Feige et al. [127] outputs a decomposition of width at most $\mathcal{O}(k\sqrt{\log k})$, where k is the optimum width. As Austrin et al. [20] show, under certain complexity assumptions no polynomial-time constant factor approximation exists.

From the point of view of some applications, for instance the bidimensionality theory that we discuss in Section 3.2.4, all three aspects of an approximation algorithm for treewidth are important: the polynomial factor, the dependence on the parameter, and the approximation ratio. For this reason the problem of computing treewidth has been revisited recently by Bodlaender et al. [46], who give an algorithm working in $\mathcal{O}(c^k\cdot n)$ time for some constant c, and outputting a tree decomposition of width at most $5k+4$. We invite to the work of Bodlaender et al. for a comprehensive overview of the known algorithms for treewidth.

3.2.2 Treewidth and $\mathbf{MSO}_2$

Let us start with recalling the famous Courcelle's theorem.

Theorem 16 (Courcelle, [79]). *There exists an algorithm that, given an n-vertex graph G of treewidth t and a formula φ of $\mathbf{MSO}_2$, checks if φ is satisfied in G in time $f(||\varphi||, t) \cdot n$ for some computable function f.*

We remark that the formula φ in Theorem 16 can have some free variables (vertex or edge, and possibly monadic) that have prescribed interpretation in G.

In the classical proof of Theorem 16 one first computes a tree decomposition $\mathcal{T}$ of G of width bounded by a function of t, using for instance the algorithm of Bodlaender [42] (the original proof assumed the tree decomposition to be given on input). Then, we encode this tree decomposition as a binary tree T over an alphabet Σ_t, whose size bounded by a function of t. Similarly, the $\mathbf{MSO}_2$ formula φ on graphs can be translated to an equivalent **MSO** formula φ' on trees over Σ_t. The finishing step is to transform φ' into a tree automaton $\mathcal{A}'$ of size bounded by a function of $||\varphi'||$, and run it on T.

As usual with relations between **MSO** and automata, the size of the obtained automaton may depend non-elementarily on $||\varphi||$. Very roughly speaking, each quantifier alternation in φ adds one level of a tower to the dependence on t. However, it must be admitted that precise tracking of the running time given by Courcelle's theorem is generally very difficult and of course depends on the used version of the proof. For these reasons, algorithms given by Courcelle's theorem are generally regarded as inefficient, and to provide precise bounds on the running time for a particular problem one usually needs to design an explicit dynamic program by hand. However, there are a few attempts of implementing Courcelle's theorem efficiently [219, 220].

Courcelle's theorem, as stated in Theorem 16, does not imply directly tractability of many important problems. Consider for example the Vertex Cover problem. Clearly, we can express that a graph admits a vertex cover of size at most k by an $\mathbf{MSO}_2$ formula of length $\mathcal{O}(k^2)$, but this gives us only an algorithm with running time $f(k, t) \cdot n$, instead of $f(t) \cdot n$; recall that the standard dynamic programming routine for Vertex Cover works in time $\mathcal{O}(2^t \cdot t^{\mathcal{O}(1)} \cdot n)$, which is independent of the target cardinality k. For this reason, the original framework of Courcelle has been extended by Arnborg et al. [16] to include also optimization problems; we remark that a very similar result was obtained independently by Borie et al. [53]. In the framework of Arnborg et al. we are given an $\mathbf{MSO}_2$ formula $\varphi(X_1, X_2, \ldots, X_q)$ with $X_1, X_2, \ldots, X_q$ being free monadic variables, and a linear combination $\alpha(|X_1|, |X_2|, \ldots, |X_q|)$ of cardinalities of these sets. The result is that (i) one can in $f(||\varphi||, t) \cdot n$ time optimize (minimize or maximize) the value of $\alpha(|X_1|, |X_2|, \ldots, |X_q|)$ for sets $X_1, X_2, \ldots, X_q$ for which φ is satisfied, (ii) in time $f(||\varphi||, t) \cdot n^{\mathcal{O}(1)}$ ask whether there exist sets $X_1, X_2, \ldots, X_q$ satisfying φ with a precise value of $\alpha(|X_1|, |X_2|, \ldots, |X_q|)$. Thus, we can express finding minimum size vertex cover by taking a constant-size formula $\varphi(X)$ that checks whether X is a vertex cover, and applying the result of Arnborg et al. for minimization.

The natural question to what extent the complexity blow-up in Courcelle's theorem is necessary has also been intensively studied. Frick and Grohe [155] have shown that, unless P=NP, no algorithm with running time $f(||\varphi||) \cdot n^{\mathcal{O}(1)}$ with elementary function f exists even for model checking **MSO** on words. Note that we can interpret a word as a path equipped with, for every symbol σ of the alphabet, a unary relation U_σ that is true in vertices corresponding to positions on which σ occurs[2]. Thus, model checking $\mathbf{MSO}_2$ on paths enriched by unary relation on vertices is not likely

[2]Formally, a formula of **MSO** on words can also use the order of positions in the word. However, the order

to admit FPT model checking algorithms with elementary dependence on the length of the input formula. Frick and Grohe also proved the same lower bound for **FO** on trees under a stronger assumption $FPT \neq AW[\star]$.

A different set of lower bounds have been given by Kreutzer and Tazari [228, 231, 232] under ETH. Intuitively, they show that if for a graph class $\mathcal{C}$ the treewidth of graphs from $\mathcal{C}$ cannot be bounded by $\log^c n$ for some universal constant c, then model checking $\mathbf{MSO}_2$ on this graph class is not likely even to be in XP; we refer to these works for precise statements of the results.

The same approach as in the proof of Courcelle's theorem also shows that for every t, the $\mathbf{MSO}_2$ theory of the class of graphs of treewidth at most t is decidable; recall that this means that it is decidable to check for a given $\mathbf{MSO}_2$ sentence φ whether there exists any graph of treewidth at most t satisfying φ. Roughly speaking, we just need to transform φ to a tree automaton $\mathcal{A}$ verifying satisfaction of φ on trees corresponding to tree decompositions of width at most t, and check non-emptiness of the language recognized by $\mathcal{A}$ intersected with the set of trees that represent valid decompositions, which is expressible in **MSO**; cf. [78]. Seese [294] has proven that the converse is also true: if a class of graphs $\mathcal{C}$ has decidable $\mathbf{MSO}_2$ theory, then the treewidth of every its member is bounded by some constant depending on the class only. We remark that the proof of Seese is based on a clever use of a WIN/WIN approach. Shortly speaking, if the treewidth of a graph from $\mathcal{C}$ is large, then by Theorem 12 it must contain a large grid minor which can be used to encode a long run of a Turing machine using an $\mathbf{MSO}_2$ sentence.

To conclude, all the aforementioned results show that treewidth as a graph parameter corresponds to tractability of the $\mathbf{MSO}_2$ logic on graphs in a well-defined and tight sense. It is notable that this uncovers an elegant link between complexity of graphs as topological structures, and complexity of model checking most natural variants of logic on them.

3.2.3 Model checking FO on sparse graph classes

As we have seen in the previous section, treewidth determines the limits of tractability of $\mathbf{MSO}_2$ on graphs. It is an immediate follow-up question, what kind of stronger tractability results can be proven when considering weaker logics such as **FO** or $\mathbf{MSO}_1$? The pursuit of the answer to this question has been particularly fruitful for **FO** and led to development of a deep theory of *sparse graphs*. We refer to the book of Nešetřil and Ossona de Mendez [258] for an introduction to the world of sparse graphs and their connections to logic.

The first breakthrough result is due to Seese [295], who showed that model checking **FO** is fixed-parameter tractable on graphs of bounded degree.

Theorem 17 (Seese, [295])**.** *There exists an algorithm that, given an n-vertex graph G of maximum degree d and a formula φ of* **FO***, checks if φ is satisfied in G in time $f(||\varphi||, d) \cdot n$ for some computable function f.*

Let us now shortly discuss the proof of Theorem 17 (in the form presented by Frick and Grohe [154]), since its understanding will be important in our further explanations.

First, using *locality* of **FO** we transform φ into an equivalent formula φ' which is a boolean combination of *basic local sentences*. This transformation has been formalized earlier by Gaifman [156]; however, locality of **FO** was exploited e.g. via Ehrenfeucht-Fraïse games much earlier, and in fact

relation can be easily expressed in **MSO** using the successor relation.

the original proof of Seese used a slightly different form of locality. A basic local sentence has the following form:

$$\exists_{x_1}\exists_{x_2}\ldots\exists_{x_s}\left(\bigwedge_{i<j}\mathrm{dist}(x_i,x_j)>2r\ \wedge\ \bigwedge_{i}\psi^r(x_i)\right).$$

Here, $\mathrm{dist}(\cdot,\cdot)>2r$ denotes an **FO** formula that checks whether the distance in G between vertices x_i and x_j is larger than some constant $2r$, while $\psi^r(\cdot)$ is an **FO** formula working on *r-neighborhood* of its argument, that is, on the subgraph induced by vertices in distance at most r from the argument. Informally speaking, a basic local sentence postulates existence of a number of vertices in the graph such that (i) these vertices are in at least a constant distance apart from each other, and (ii) a constant-radius neighborhood of each vertex satisfies some **FO**-expressible property.

Let us concentrate on one basic local sentence α appearing in φ'. Observe that in a graph of maximum degree at most d the r-neighborhoods of vertices are of size at most $(r+1)\cdot d^r$, i.e., of size depending on d and r only. Therefore, in $f(r,d)\cdot n$ time we can check in which vertices of G formula ψ^r is satisfied by employing a brute-force check on r-neighborhood of every vertex. Thus, to check if α is true we need to solve some sort of a constrained version of the $2r$-Scattered Set problem. Namely, given graph G of maximum degree d, a subset of vertices X (these are vertices satisfying ψ^r), and integers r,s, we ask for existence of a subset $Y\subseteq X$ of size s such that each two elements of Y are in distance more than $2r$. This problem fortunately turns out to be fixed-parameter tractable when parameterized by r, s, and d. Therefore, we can verify satisfaction of each basic local sentence appearing in φ' in FPT time, and use these values to verify satisfaction of φ'.

Observe that the argument for checking for which vertices formula ψ^r is satisfied seems like a clear overshot. We in fact employed a brute-force check on each r-neighborhood using the fact that it is of bounded size. Probably, using a more clever argument here we could strenghten the result. For instance, if we assume that the r-neighborhood of each vertex has treewidth bounded by a function of r only, then we can employ a dynamic programming routine given by the Courcelle's theorem instead. This observation, due to Frick and Grohe [154], leads to the definition of *bounded local treewidth*: a class of graphs $\mathcal{C}$ has bounded local treewidth if r-neighborhood of each vertex has treewidth bounded by $f(r)$, for some function f. Thus, Frick and Grohe [154] have proven that Theorem 17 can be lifted to any class of graphs of bounded local treewidth. Note here that one needs to lift also the algorithm for the used variant of $2r$-Scattered Set; however, as Frick and Grohe observe, the assumption of having bounded local treewidth is sufficient for this as well. The classes of graphs that have bounded local treewidth include for instance planar graphs or graphs of bounded genus.

Clearly, the natural follow-up question is about H-minor free graphs: even though they do not have bounded local treewidth because of possible existence of apices, the Decomposition Theorem gives a hope that the results can be lifted. This is indeed true and has been proven by Flum and Grohe [131]. Further improvements have been given by Dawar et al. [99] for the class of graphs with *locally excluded minors*, and by Dvořák et al. [121] and by Grohe and Kreutzer [176] for graphs of *bounded expansion*; we refer to these works for appropriate definitions. Recent research in the topic focuses on the class of *nowhere dense* graphs. It appears that many important **FO**-definable problems are indeed tractable in this class of graphs [100], and there is hope that fixed-parameter tractability of model checking the whole **FO** can be lifted there as well. This statement has been recently claimed by Dawar and Kreutzer [229]; however, unfortunately the claim needed to be retracted due to a flaw in the proof [229]. To the best of author's knowledge, the question of

fixed-parameter tractability of model checking **FO** on nowhere-dense graphs is still open.

3.2.4 WIN/WIN approaches

Exploiting a grid minor

The Excluded Grid Minor Theorem, i.e. Theorem 12, is a very convenient tool for designing FPT algorithms via WIN/WIN approaches. Consider, for instance, the classical Feedback Vertex Set problem: given a graph G and an integer k, determine if at most k vertices can be deleted from G to obtain a forest. We employ the following approach. Given a graph G and an integer k, we first run, say, the 5-approximation algorithm for treewidth of Bodlaender et al. [46] for parameter $g(k) = r\left(2\left\lceil (k+1)^{1/2}\right\rceil\right)$, where r is the function given by Theorem 12. This algorithm works in $2^{\mathcal{O}(g(k))} \cdot n$ time, and either provides a tree decomposition of G of width at most $5g(k)+4$, or correctly concludes that $\mathbf{tw}(G) > g(k)$.

If a decomposition is returned, then we may run a dynamic programming routine for Feedback Vertex Set. The straightforward approach gives a dynamic program with running time $2^{\mathcal{O}(t \log t)} \cdot n$ on graphs of treewidth t; however, a faster algorithm with running time $2^{\mathcal{O}(t)} \cdot n$ has been recently given by Bodlaender et al. [43]. Thus, we may solve the problem completely in total time $2^{\mathcal{O}(g(k))} \cdot n$.

On the other hand, if the approximation algorithm concluded that $\mathbf{tw}(G) > g(k)$, then by Theorem 12 we can conclude that G contains a $\left(2\left\lceil (k+1)^{1/2}\right\rceil\right) \times \left(2\left\lceil (k+1)^{1/2}\right\rceil\right)$ grid minor. However, it is easy to see that such a minor contains $\left(\left\lceil (k+1)^{1/2}\right\rceil\right)^2 \geq k+1$ vertex-disjoint cycles; see for example Figure 3.1. Observe that any feedback vertex set of G must necessarily contain at least one vertex from each of these cycles, which implies that no feedback vertex set of size at most k exists and we may safely provide a negative answer. Note here that in order to perform this reasoning we do not need a constructive proof of Theorem 12: there is no need to construct the actual grid to provide a negative answer to the problem, a proof of its existence is sufficient.

This meta-framework may be used to settle fixed-parameter tractability of a number of classical problems; examples include Vertex Cover, k-Path or Packing Vertex-Disjoint Cycles. For the technique to work we need the following property of the problem: existence of a large grid minor is a certificate of a trivial YES- or a trivial NO-answer. However, the framework usually can be used only for classification purposes, as the obtained algorithms are far from being optimal. For instance, in the Feedback Vertex Set example we obtain a $2^{\mathcal{O}(k^{\mathcal{O}(1)})} \cdot n$ algorithm for a large polynomial in the exponent, and the complexity is not double-exponential only because of usage of the new proof of the Excluded Grid Minor Theorem of Chekuri and Chuzhoy [63]. However, the Feedback Vertex Set problem may be solved as fast as in $\mathcal{O}^*(3^k)$ randomized time [91] or $\mathcal{O}^*(3.592^k)$ deterministic time [221].

Bidimensionality

The technique becomes really powerful when applied to the world of planar graphs, or more generally graph classes with topological constraints. Consider exactly the same algorithm for the Feedback Vertex Set problem, but instead of the general Excluded Grid Minor Theorem we use the version for planar graphs that has linear function r. Then, the algorithm works in $2^{\mathcal{O}(\sqrt{k})} \cdot n$ time, that is, in subexponential parameterized time!

This observation was probably first noted by Fomin and Thilikos [148]. However, it should be remarked that existence of subexponential parameterized algorithms for problems on planar graphs

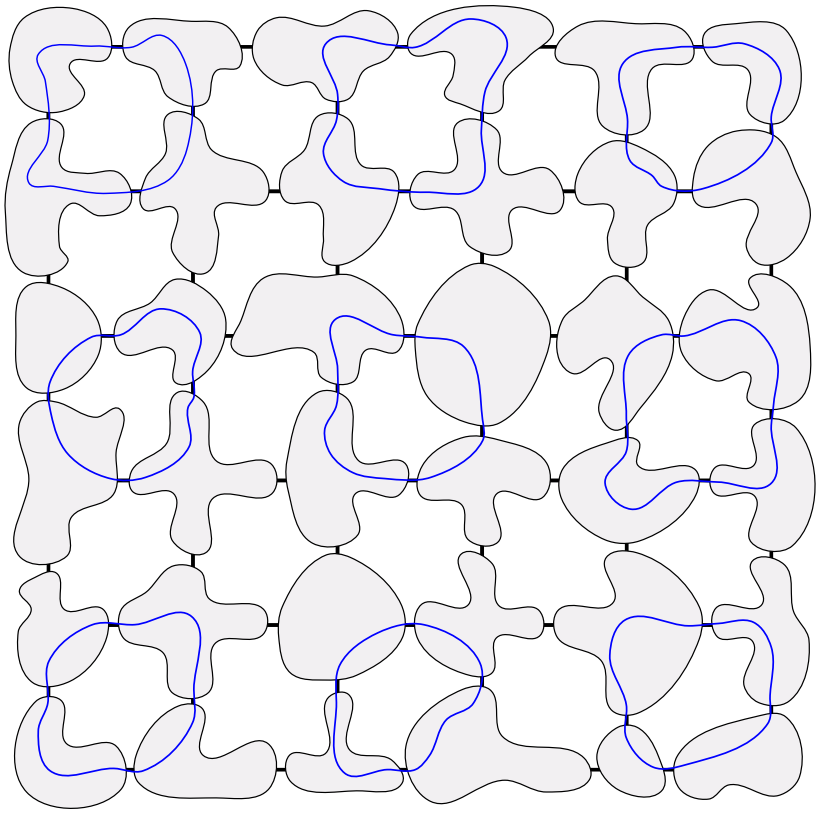

Figure 3.1: A 6×6 grid minor and 9 vertex-disjoint cycles in it.

was already observed earlier by Alber et al. [6, 7] using a much more involved layering technique resembling the classical Baker's approach. Notably, the approach can be also applied to problems for which the framework on general graphs fails. Consider for instance the Dominating Set problem. While the problem is $W[2]$-hard on general graphs, it is easy to see that for planar graphs existence of a $(3\lceil k+1\rceil)\times(3\lceil k+1\rceil)$ grid minor implies existence of a 2-*scattered set* of size at least $k+1$, i.e. a set of vertices in distance at least 2 from each other. Such a scattered set, however, certifies that any dominating set must be of size at least $k+1$. Note that such a reasoning breaks in the general case, as the whole grid minor can be dominated by one vertex. Thus, while existence of an FPT algorithm for the Dominating Set problem is implausible on general graphs, by applying the same technique as for Feedback Vertex Set we obtain an algorithm for planar graphs with running time $2^{\mathcal{O}(\sqrt{k})}\cdot n$.

The name *bidimensionality* was probably first used by Demaine et al. [104], who put the approach into a formalism of more general nature, and also extended many of the results to the bounded-genus and H-minor-free settings, thus expanding greatly capabilities of the technique. In the sense of Demaine et al., a graph parameter must be *bidimensional* to admit application of the technique. Roughly speaking, this means that existence of a grid of size linear in the parameter must be a certificate of a YES- or a NO-answer. The technique was later generalized also for some problems not falling directly under the regime of the proposed formalism, for instance in the context of partial covering problems [141], or in the context of other geometric graph classes [103, 142].

It appears that the insights given by the bidimensionality theory are useful also in other fields. Of particular significance are applications in kernelization [49, 138, 143, 144], where the tools of

bidimensionality led to development of the *meta-kernelization* framework of Bodlaender et al. [49], further refined by Fomin et al. [143]: a powerful algorithmic meta-technique that explains existence of many linear kernels on bounded genus graphs and H-minor free graphs by expressibility in $\mathbf{MSO}_2$ and further technical conditions related to bidimensionality. Bidimensionality is also useful in the context of approximation, in particular it provides tools for designing PTASes and EPTASes for many problems on planar, bounded genus, and H-minor free graphs [105, 140].

We refer to surveys of Dorn et al. [118] and of Demaine and Hajiaghayi [106] for more information on bidimensionality.

Other examples

Apart from the examples mentioned above, the Excluded Grid Minor Theorem has found other multiple applications. Many of them are related to topological problems and mimic the WIN/WIN approach used by Robertson and Seymour in the algorithm for minor testing. Namely, we aim at identifying a large, flat, grid-like structure such that (almost) no vertices in the interior of the structure are adjacent to anything outside. If such a structure can be found, we design an irrelevant vertex (edge) rule: we prove that some hypothetical solution of the problem can be assumed not to use one of the middle vertices (edges) of the structure. Hence, this vertex (edge) may be safely deleted, and we can restart the whole algorithm. If no grid-like structure can be found, then the treewidth of the graph is small and we may employ a dynamic programming routine.

Since the arguments behind irrelevant vertex rules are usually very technical and delicate, in order to have a better control of the situation one usually uses more refined versions of obstructions instead of original grid minors. In many cases it is convenient to work with a *wall*; instead of giving the definition of a wall formally, we hope that the reader will understand this notion by examining Figure 3.2. It is easy to see that if G admits a $k \times k$ grid minor, then it admits also a wall of height and width linear in k as a topological subgraph. Conversely, a wall of height and width linear in k admits a $k \times k$ grid as a minor. Therefore, the problems of finding large grid minors and large subdivisions of walls are almost equivalent. However, the assumption that the wall is actually a topological subgraph gives us more control over its combinatorial properties. Note, for instance, that if a subdivision of a wall is a subgraph of a planar graph, then every vertex of the wall not lying on its perimeter can be adjacent only to vertices embedded into the three neighboring bricks, and in particular it cannot be adjacent to any vertex embedded into the outer face of the wall. This 'flatness' condition can be generalized also to general graphs via the concept of a *flat wall*, see e.g. [175, 285], which is a crucial notion in the minor testing algorithm of Robertson and Seymour [285].

Examples of results that can be obtained via finding irrelevant objects in large grid-like structures include: FPT algorithms for obtaining a planar graph using at most k vertex deletions [206, 253]; FPT algorithms for computing the crossing number of a graph [173, 209]; a polynomial-time algorithm for finding induced paths of given parity between a pair of vertices in a planar graph [200]; an FPT algorithm for checking whether a bounded genus graph G can be contracted to a small graph H, whose size is the parameter [201]; a double-exponential FPT algorithm for Vertex Disjoint Paths on planar graphs [2]; and decidability of Hadwiger's conjecture [210], among many others.

3.3 Other undirected width measures

We now discuss other width measures of undirected graphs. We start with cliquewidth and rankwidth that measure algebraic complexity of a graph rather than topological. Then we proceed

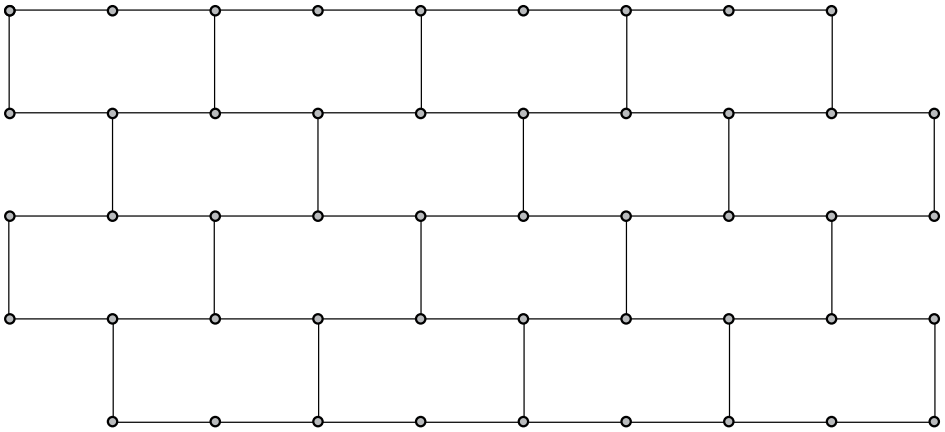

Figure 3.2: A wall of height 4 and width 4.

to close relatives of treewidth, namely branchwidth and carving-width. Finally, we discuss width measures based on linear orderings of vertices, in particular cutwidth.

3.3.1 Cliquewidth and rankwidth

Cliquewidth has been introduced by Courcelle and Olariu [83] as a parameter that corresponds to $\mathbf{MSO}_1$ in the same manner as treewidth corresponds to $\mathbf{MSO}_2$. Cliquewidth of a graph G is upper-bounded by $3 \cdot 2^{\mathbf{tw}(G)-1}$ [77]; however, there are classes of graphs of constant cliquewidth and unbounded treewidth, e.g. cliques. Intuitively, graphs of cliquewidth at most k may be constructed in a tree-like manner formalized in the concept of a *clique expression*: at each point of the construction the graph may be partitioned into at most k groups of vertices, such that vertices in one group are already indistinguishable and will have the same neighborhood in the part of the graph that is not yet introduced. We refer to Section 2.3.3 for a formal definition.

Similarly to treewidth, also graphs of bounded cliquewidth admit simple dynamic programming routines. For instance, on graphs of cliquewidth at most k one can solve the Independent Set problem in $\mathcal{O}(2^k \cdot k^{\mathcal{O}(1)} \cdot (n+m))$ time [185] and the Dominating Set problem in $\mathcal{O}(4^k \cdot k^{\mathcal{O}(1)} \cdot (n+m))$ time [52]. We remark that in these results one assumes that the graph is given together with an appropriate clique expression constructing it. The following meta-result of Courcelle et al. [82] explains existence of these dynamic programs via expressibility in $\mathbf{MSO}_1$, and is a direct analogue of Courcelle's theorem for treewidth (Theorem 16).

Theorem 18 ([82])**.** *There exists an algorithm that, given a graph G of cliquewidth at most k together with the corresponding clique expression constructing it, and a formula φ of $\mathbf{MSO}_1$, checks if φ is satisfied in G in time $f(||\varphi||, k) \cdot (n+m)$ for some computable function f, where $n = |V(G)|$ and $m = |E(G)|$.*

Again, formula φ in Theorem 18 can have some free vertex variables (possibly monadic) with prescribed interpretations in G. Similarly as in the results of Arnborg et al. [16], also the result of Theorem 18 can be generalized to encompass also optimization problems, and in particular the aforementioned examples of the Independent Set and Dominating Set problems. We note that a version of Theorem 18 for directed graphs will be used in Part II of this thesis.

The definition of cliquewidth, as introduced in [83], is mostly inspired by the earlier work of Courcelle et al. on graph rewriting grammars [80, 81]. Therefore, while very well-suited for designing dynamic programming routines, the combinatorics of cliquewidth turned out to be extremely difficult to handle, in particular when designing exact or approximation algorithms is concerned. In fact, NP-hardness of computing cliquewidth exactly has been established only a few years after introducing this parameter [129], and it is still unknown whether computing cliquewidth exactly can be done even in XP time. However, in order to remove from Theorem 18 the inconvenient requirement of providing an explicit clique expression that constructs the graph, it is sufficient just to design an FPT approximation algorithm for cliquewidth.

This gap has been bridged by Oum and Seymour [265], who define a new parameter *rankwidth* that can be viewed as a reformulation of cliquewidth suited for computing its value. Intuitively, the main idea of rankwidth is to measure diversity of neighborhoods in terms of the rank of a certain adjacency matrix (see Section 2.3.4 for an appropriate definition), and exploit submodularity of the rank function algorithmically. For any graph G it holds that $\mathbf{rw}(G) \leq \mathbf{cw}(G) \leq 2^{\mathbf{rw}(G)+1} - 1$ [265] and a suitable clique expression may be constructed from the corresponding rank decomposition of G. Therefore, in order to provide an FPT approximation algorithm for cliquewidth it suffices to provide an FPT approximation algorithm for rankwidth. Such an algorithm has been presented by Oum and Seymour [265] in their work. The algorithm works in $2^{\mathcal{O}(k)} \cdot n^9 \log n$ time[3] and either provides a decomposition of width at most $3k+1$ or correctly concludes that $\mathbf{rw}(G) > k$. The polynomial factor and the approximation ratio of the algorithm has been later improved to n^3 and $3k-1$, respectively [262]. An exact FPT algorithm for rankwidth has been given by Hliněný and Oum [189].

It is also noteworthy that the algorithms designed for rankwidth can be also used to compute branchwidth of matroids [189, 265]. On the other hand, it appears that rankwidth and cliquewidth are tightly connected to algebraic containment notions called *vertex-minors* and *pivot-minors* [261]. This leads to an elegant containment theory resembling the Graph Minors project in principles, in which, however, answers to many problems are still unknown. We refer to the work of Courcelle, Geelen, Jeong, Kwon, and Oum [84, 164, 197, 261, 263, 264] for more details on this theory. We also refer to a recent survey of Hliněný et al. [190] for a comprehensive introduction to cliquewidth, rankwidth, and related width measures.

3.3.2 Branchwidth and carving-width

Branchwidth has been introduced by Robertson and Seymour [284] as an alternative definition of treewidth, for which some of the reasonings are cleaner and more concise; we refer to Section 2.3.4 for a formal definition. Branchwidth and treewidth are bounded by linear functions of each other; more precisely, $\mathbf{bw}(G) - 1 \leq \mathbf{tw}(G) \leq \frac{3}{2}\mathbf{bw}(G) - 1$ for any graph G of branchwidth more than 1 [284]. While branchwidth can be smaller than treewidth, for many problems one can implement dynamic programming routines working on a branch decomposition with only slightly larger base of the exponent in the running time. For example, while the classical dynamic program on tree decomposition for Independent Set works in $\mathcal{O}(2^t \cdot t^{\mathcal{O}(1)} \cdot n)$ time, where t is the width of given tree decomposition, for branch decomposition of width k one can devise an algorithm with running time $\mathcal{O}(2.28^k \cdot n)$ via fast matrix multiplication [117]. As Fomin and Thilikos argue [148], branchwidth is probably a better-suited version of treewidth in the context of planar graphs. In fact, somehow

[3]The dependence on the parameter has not been provided in the work of Oum and Seymour; however, a careful examination of the algorithm yields a $2^{\mathcal{O}(k)}$ bound on this factor.

surprisingly branchwidth of a planar graph can be computed in polynomial time [298], while the classical complexity of computing treewidth of a planar graph remains a notorious open problem.

Carving-width, introduced by Seymour and Thomas [298], is a similar concept to branchwidth, but one defines it in a seemingly more natural way by partitioning the vertex set instead of edge set, and measuring the width by the number of edges cut instead of vertices; see Section 2.3.4 for a formal definition. While treewidth may be easily bounded by a linear function of carving-width, carving-width of even as simple graphs as stars is unbounded. For this reason it may be argued that carving-width can behave counter-intuitively and does not correspond to any structural properties of the graph. It can be however useful as an auxiliary parameter, for instance in the context of branchwidth [298].

3.3.3 Pathwidth and cutwidth

Pathwidth is a linearized version of treewidth introduced in the Graph Minors project [279], where we require the tree decomposition to be a path instead of an arbitrary tree. While pathwidth of a graph is of course at least its treewidth, it can be unbounded even on trees. Pathwidth can be computed exactly by an FPT algorithm in time $f(p) \cdot n$ [42, 51] for some computable function f, where p is the optimum value of pathwidth.

Dynamic programming routines working on tree decompositions can be of course translated to path decompositions, and in some cases the base of the exponent can be made smaller. The reason is that in path decompositions one does not need to consider the join operation; this operation is usually the most expensive one when tree decompositions are concerned, and makes the dynamic program work in quadratic time in terms of the number of states, instead of linear. It has been observed only recently by van Rooij et al. [305] that in most cases one can use fast subset convolutions to speed-up the join operation, thus making the bases of exponents for many important problems equal for path decompositions and tree decompositions. However, for some algorithms it is still not known how to speed-up the join operation using convolutions, with a notable example of the currently fastest algorithm for Hamiltonian Cycle [88].

Pathwidth has also quite unexpected applications in the field of exact algorithms. Fomin and Høie [135] have proved that pathwidth of an n-vertex graph with maximum degree 3 (i.e., a subcubic graph) is at most $(\frac{1}{6} + o(1))n$. Pipelining this result with dynamic programming on path decomposition gives best known exact algorithms on subcubic graphs for many important problems, including Dominating Set [135] and Hamiltonian Cycle [88].

Cutwidth, in turn, is an exemplary member of the family of *graph layout* parameters, where we would like to order the vertices of a graph to minimize some cost function. In case of cutwidth this function is defined as naturally as the maximum number of edges between any prefix of the ordering and the corresponding suffix; see Section 2.3.5 for a formal definition. Cutwidth has been introduced already in 1973 by Adolphson and Hu [3]. Thilikos et al. [302] have presented a linear-time FPT algorithm that computes it exactly in $f(c) \cdot n$ time for some function f, where c is the optimal value of cutwidth. The linearity of the definition can be used to approach cutwidth and related width measures via automata, as shown by Bodlaender et al. [48]. We refer to the surveys of Serna and Thilikos [296] and of Diaz et al. [111] for more information on other graph layout parameters.

3.4 Width measures of directed graphs

Due to importance of the concept of treewidth, there were several attempts to find its natural analogue in the directed setting [30, 31, 194, 198, 292]; however, none of the proposed notions shares all the nice properties of undirected treewidth. In this section we briefly survey three most prominent propositions, namely directed treewidth, DAG-width and Kelly-width. We also mention the existing work on directed pathwidth in general digraphs, since this width notion will be crucial in our reasonings for semi-complete digraphs in Part II. Finally, we survey results about implausibility of existence of algorithmically and structurally interesting width measures of digraphs.

3.4.1 Directed treewidth

Historically the first, directed treewidth was introduced by Johnson et al. [198] as a generalization of the undirected notion contrived to solving linkage problems like Vertex Disjoint Paths or Hamiltonian Cycle; see also a short introductory note by Reed [276]. Similarly as in the undirected setting, directed treewidth is defined via decompositions, called in this case *arboreal decompositions*. An arboreal decomposition $\mathcal{R}$ of a digraph D is an outbranching R (a rooted tree with all the edges directed away from the root), where each node $x \in V(R)$ has associated a set of vertices $W_x \subseteq V(D)$ and every arc $a \in E(R)$ has associated a set of vertices $X_a \subseteq V(D)$. We require that $(W_x)_{x \in V(R)}$ is a partition of $V(D)$ into nonempty subsets. The separation property of undirected treewidth is translated as follows. Take any arc $a \in E(R)$ and let V_a be the union of bags W_x associated with nodes $x \in V(R)$ that have a on the unique directed paths connecting them to the root of R. Then we require that there is no directed walk in $D \setminus X_a$ that starts and ends in $V_a \setminus X_a$ and which traverses a vertex belonging to $V(D) \setminus (V_a \cup X_a)$. The *width* of the decomposition is defined as $\max_{x \in V(D)} |W_x \cup \bigcup_{a \sim x} X_a| - 1$ where $\sim$ stands for the relation of incidence, and the directed treewidth is the minimum possible width among all arboreal decompositions.

Johnson et al. prove a number of structural results about directed treewidth. In particular, they relate it to the notion of *havens*, which are obstacles for admitting an arboreal decomposition of small width. More precisely, existence of a haven of order w certifies that directed treewidth is at least $w - 1$, while every digraph with directed treewidth at least $3w - 1$ has a haven of order w. The proof of this theorem can be turned into an approximation algorithm for directed treewidth working in XP time. However, havens are canonical obstacles of similar flavor as tangles in the undirected setting, and therefore do not have form of embedded combinatorial objects such as grid minors. Since having obstacles of the second type is much more useful from the point of view of algorithmic applications, for example for applying the irrelevant vertex technique, it is important to find an analogue of the grid minor for directed treewidth as well. Johnson et al. conjectured that such an obstacle is a directed grid, depicted on Figure 3.3. More precisely, the conjecture is that there exists a function $r(\cdot)$ such that any digraph of directed treewidth at least $r(w)$ contains a directed grid with w cycles and $2w$ paths crossing the cycles as a *butterfly minor* (see [198] or [1] for a formal definition of this containment notion). Johnson et al. claim that the conjecture holds for planar graphs; however, the proof has never been published. Very recently, Kawarabayashi and Kreutzer [227] have announced that they are able to prove the conjecture for the class of digraphs whose underlying undirected graph excludes H as a minor, for any fixed graph H; note that this generalizes the planar case. The general case still remains open.

Johnson et al. have proved that the Hamiltonian Cycle problem is in XP when parameterized by directed treewidth, while Vertex Disjoint Paths is in XP when parameterized by directed

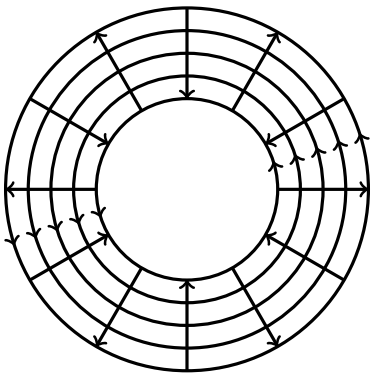

Figure 3.3: A directed grid — the conjectured obstacle for small directed treewidth.

treewidth and the number of terminal pairs. Both these results are essentially tight. Lampis et al. [236] have shown that the Hamiltonian Cycle problem is $W[2]$-hard when parameterized by directed treewidth, while Slivkins [300] has shown that the Vertex (Edge) Disjoint Paths problems are $W[1]$-hard when parameterized by the number of terminal pairs even on DAGs, which have directed treewidth 0. The latter lower bound suggests that directed treewidth is not likely to be helpful in devising FPT algorithms for topological problems in directed graphs — XP running time is the limit. Moreover, directed treewidth does not enjoy such general meta-theorems as the Courcelle's theorem (Theorem 16) or its optimization version of Arnborg et al. [16]. In fact, as Ganian et al. [158] and Kreutzer and Ordyniak [230] show, many important problems remain NP-hard on digraphs of constant directed treewidth, with notable examples of Feedback Arc (Vertex) Set [230] and Directed Dominating Set [158]. The latter result holds even for DAGs.

3.4.2 DAG-width and Kelly-width

DAG-width has been introduced independently by Berwanger et al. [29] and by Obdržálek [259] as an alternative to directed treewidth; we refer to the joint journal version [30] of these works for a comprehensive introduction. The idea is to take as the starting point of generalization not the concept of a tree decomposition, but the alternative definition of treewidth via *graph searching games* that dates back to the work of Seymour and Thomas [297].

Consider the following game on an undirected graph G, played by k cops equipped with choppers and a robber. First, the cops place themselves in vertices of G and the robber places herself in any unoccupied vertex. The cops and the robber move in turns and the cops always see where the robber is. During each turn, any subset of cops can lift in their choppers declaring vertices where they will land. While the lifted cops are in the air, the robber can move along any path in the graph, providing she does not pass through any vertex occupied by a cop that has not lifted. Finally, the lifted cops land and a new turn begins. The cops win by eventually landing on a vertex occupied by the robber, while the robber wins by evading the cops infinitely. As Seymour and Thomas [297] show, the minimum number of cops for which a winning strategy for cops exists is equal to $\mathbf{tw}(G)+1$.

By altering the game rules, for example visibility of the robber or constraints on cops'/robber's moves, we obtain different games that correspond to different width parameters. The idea of DAG-width is to consider the original game for treewidth of Seymour and Thomas on digraphs, and to request the robber to respect the arc directions. However, the definition also assumes a technical

condition of *monotonicity* of the cops' strategy, that is, that by following the strategy the cops never land on a vertex twice. While in the undirected setting the monotone and non-monotone strategies are equivalent [297] (if k cops have a strategy to catch the robber, then they have also a monotone one), this is already not true in the directed setting, as shown by Kreutzer and Ordyniak [230].

DAG-width admits also a definition via a decomposition, where the idea is to use a DAG instead of an outbranching as the decomposition's shape. As the directed treewidth of a digraph of DAG-width k is bounded by $3k + 1$ [30], DAG-width inherits all the tractability results from directed treewidth; however, there exist families of digraphs with constant directed treewidth and unbounded DAG-width [30]. Moreover, Berwanger et al. [30] have shown that the Parity Games problem is in XP when parameterized by the DAG-width; in fact, showing a tractability result for Parity Games was the main motivation of [29]. In addition, DAG-width can be computed exactly in XP time [30].

Kelly-width, introduced by Hunter and Kreutzer [194], is a concept similar to DAG-width, but here the starting point of the generalization is a relation between treewidth and chordal graphs via elimination orderings; we refer to the work Hunter and Kreutzer for a formal definition that is somewhat technical. Kelly-width admits also alternative definitions via a version of cops-and-robber game and via a form of a decomposition that is similar to the one used for DAG-width. The main advantage of Kelly-width is the corresponding decomposition has linear size with respect to the decomposed digraph, which is not the case for DAG-width. In fact, for DAG-width we do not have even a polynomial bound on the size of the decomposition, and hence it is unclear whether computing DAG-width is in NP [30]. On the other hand, it is still not known whether computing Kelly-width can be done in XP time [194].

The directed treewidth of a digraph of Kelly-width k is bounded by $6k - 2$, while there exist families of digraphs of constant directed treewidth and unbounded Kelly-width [194]. An interesting open question, formulated both in [194] and in [30], is to find relation between DAG-width and Kelly-width. The authors conjecture that the two width measures are within a constant multiplicative factor from each other.

Despite being possibly much larger than directed treewidth, both DAG-width and Kelly-width do not admit many more tractability results. In fact, the negative results of Lampis et al. [236] (W[2]-hardness of Hamiltonian Cycle parameterized by directed treewidth) and of Kreutzer and Ordyniak [230] (NP-hardness of Feedback Arc (Vertex) Set and other problems on digraphs of constant directed treewidth) can be lifted to parameterizations by DAG-width and Kelly-width. Moreover, since DAGs have DAG-width and Kelly-width equal to 1, the same holds for the negative results of Slivkins [300] on the Vertex (Edge) Disjoint Paths problems, and of Ganian et al. [158] for Directed Dominating Set and other problems.

3.4.3 Directed pathwidth

Directed pathwidth has been introduced in the 90s by Reed, Seymour, and Thomas as a natural generalization of the undirected notion. Similarly to its undirected analogue, directed pathwidth is defined via a notion of a *path decomposition*. Again, a path decomposition is a sequence of bags, but this time we require that every arc of the digraph is either contained in some bag, or is directed from some bag appearing later in the decomposition to an earlier one. In Section 5.2.3 we give a formal definition of this notion, and provide a number of simple observations.

A digraph of directed pathwidth k has DAG-width at most $k + 1$ [30], and a similar bound can be easily obtained for Kelly-width. Thus, both Kelly-width and DAG-width are sandwiched between directed treewidth and directed pathwidth. Unfortunately, this does not mean that pa-

rameterization by directed pathwidth gives more tractability results; the aforementioned negative results [158, 236, 300] actually hold also when using directed pathwidth as the parameter.

A version of cops-and-robber game for directed pathwidth has been studied by Bárat [25]. Moreover, an XP exact algorithm for directed pathwidth has been recently proposed by Tamaki [301]. To the best of author's knowledge, it remains unknown whether directed pathwidth of a general digraph can be computed in fixed-parameter tractable time.

3.4.4 No good width measures for directed graphs?

The lower bounds given in the previous sections show that none of the proposed width measures of digraphs enjoys even a fraction of algorithmic properties of undirected treewidth. Moreover, it is also unclear how the proposed width measures behave with respect to different containment notions. For instance, Adler [1] has shown that taking a butterfly minor can even increase the directed treewidth of a graph; recall that butterfly minors were used to state the conjecture about existence of a large directed grid in a digraph with large directed treewidth. In addition, natural cops-and-robber games on digraphs behave counter-intuitively: in many cases non-monotone strategies are provably stronger than monotone ones [1, 230].

Therefore, it is natural to ask whether there exists any sensible width notion that is both algorithmically useful and has good structural properties, meaning that it is closed under some form of minor relation. This question has been systematically investigated by Ganian et al. [159], who propose the following formalizations of expressions 'sensible', 'algorithmically useful' and 'good structural properties'. First, for being algorithmically useful Ganian et al. request that verifying satisfiability of any sentence of $\mathbf{MSO}_1$ logic should be in XP when parameterized by the considered width measure; note that this is a relaxation of results like Courcelle's theorem. Second, for good structural properties they request that the considered width measure is closed under taking *directed topological minors*; the definition of this containment relation is somewhat technical, however the authors argue that it naturally corresponds to cops-and-robber games. Third, for sensibility Ganian et al. need one more technical condition they call *efficient orientability*. Shortly speaking, a width measure δ is efficiently orientable if given an undirected graph G one can in polynomial time find an $f(OPT)$-approximation of the orientation of edges of G that yields a digraph of the minimum possible value of δ. Both DAG-width and Kelly-width are efficiently orientable.

Ganian et al. have proven a somewhat surprising result that requesting the three aforementioned conditions already implies that the considered width measure must be bounded by a function of treewidth of the underlying undirected graph, unless P=NP. Intuitively speaking, this means that there is no width notion for general directed graphs that both shares good properties of undirected treewidth and does not lose information about orientation of arcs.

Chapter 4

The optimality program

The goal of the optimality program is to pursue tight bounds on the complexity of parameterized problems, focusing particularly on dependence on the parameter. In the literature one can find a full variety of functions $f(k)$ in the running times of FPT algorithms: from non-elementary given by the Graph Minors theory or Courcelle's theorem, through multiple-exponential and single exponential for many classical problems, to even subexponential as in the case of bidimensionality. The question of qualitative differences between problems with different parameterized complexity is of main interest of the optimality program. The thesis is that obtaining a tight lower bound for a parameterized problem not only proves that further efforts of designing faster algorithms are probably futile, but also gives new insights into the problem's structure by extracting and exhibiting the source of intractability.

The formulation of ETH and SETH by Impagliazzo and Paturi [195] (see Section 2.2.3 for formal definitions) provided very convenient complexity assumptions for proving lower bounds that almost match known upper bounds for many problems. Since then the optimality program became a thriving subbranch of parameterized complexity, defined by Marx [252] as one of the key future directions in the field. For reference we refer to the survey of Lokshtanov et al. [240] and an appropriate section of the survey of Marx [252], of which the following chapter is an updated compilation.

4.1 Results following immediately from ETH

4.1.1 Classical complexity

Essentially, it has been observed already by Impagliazzo et al. [196] that assuming ETH one can exclude existence of algorithms that are subexponential in terms of instance size for a number of classical NP-hard problems[1]. More precisely, Impagliazzo et al. consider a syntactic class of problems SNP (for a precise definition we refer to [196]) and showed that 3-CNF-SAT is complete for SNP with respect to a form of reductions (called SERF reductions) that preserve existence of subexponential algorithms. In particular, any linear NP-hardness reduction from 3-CNF-SAT, i.e., a reduction that takes an instance of 3-CNF-SAT with n variables and m clauses and produces an equivalent instance of the target problem L of size $\mathcal{O}(n+m)$, is a SERF reduction. Indeed, it can be easily seen that pipelining such a reduction with the assumed subexponential algorithm

[1] ETH has been formulated only in the later work of Impagliazzo and Paturi [195], but this observation is already present in [196].

for L yields an algorithm for 3-CNF-SAT that works in $2^{o(n+m)}$ time, thus contradicting ETH by Corollary 5. Even though the notion of SERF reductions is more general, for most classical NP-hard problems, like Vertex Cover, Independent Set, Dominating Set, or Hamiltonian Cycle, the standard NP-hardness reductions known in the literature are in fact linear. Thus, we immediately obtain as a corollary that none of these problems admits a subexponential algorithm in terms of instance size, unless ETH fails.

Following this line of reasoning, one can examine also classical NP-hardness reductions for problems on planar graphs, like Planar Vertex Cover, Planar Dominating Set or Planar Hamiltonian Cycle. Typically, such reductions start with a concept of a linear construction that does not necessarily preserve planarity, like, for instance, an NP-hardness reduction for the problem in general graphs. Then one needs to carefully design a planar *crossover gadget* that substitutes a crossing of two edges of the preliminary construction, thus making the whole resulting instance planar; see for reference NP-hardness reductions for Planar Vertex Cover [162] or Planar Hamiltonian Cycle [163]. Thus, such a reduction takes as input an instance of 3-CNF-SAT with n variables and m clauses, and produces an instance of the target problem L of size $\mathcal{O}((n+m)^2)$: while the preliminary construction is linear, the instance size can be blown up quadratically when introducing the crossover gadgets.

Observe now that if there existed an algorithm for L with running time $2^{o(\sqrt{N})}$ where N is the total number of vertices of the graph, then pipelining the reduction with such an algorithm would yield an algorithm for 3-CNF-SAT that works in $2^{o(n+m)}$ time, again contradicting ETH by Corollary 5. As a result, neither of Planar Vertex Cover, Planar Dominating Set or Planar Hamiltonian Cycle admits an algorithm working in time $2^{o(\sqrt{N})}$, and the same result holds for many other problems on planar graphs. Note, however, that all these problems in fact do admit algorithms working in time $2^{\mathcal{O}(\sqrt{N})}$. Such algorithms can be obtained by Lipton-Tarjan separator theorem, or in more modern language, by pipelining the fact that a planar graph on N vertices has treewidth $\mathcal{O}(\sqrt{N})$ with dynamic programming on a tree decomposition. Therefore, we obtain surprisingly tight bounds on the complexity of the considered problems: while algorithms of running time $2^{\mathcal{O}(\sqrt{N})}$ exist, achieving running time $2^{o(\sqrt{N})}$ seems unlikely.

4.1.2 Parameterized complexity

Applicability of ETH to parameterized problems has been probably first noted by Cai and Juedes [55], who observe that for all the problems where the parameter is naturally bounded by the size of the instance, a lower bound on the classical complexity of the problem implies a lower bound on the parameterized complexity as well. For instance, for the Vertex Cover problem a parameterized algorithm with running time $\mathcal{O}^*(2^{o(k)})$ has also classical complexity $2^{o(N)}$ where N is the number of vertices of the graph, and we have already refuted existence of such algorithms under ETH. The same observation can be applied to other parameterized problems; Cai and Juedes in fact consider a class of problems called Max-SNP and show that an analogous corollary may be concluded for all problems that are complete for Max-SNP with respect to SERF reductions. Note that in the case of Vertex Cover this lower bound is essentially tight, as the problem can be solved by a simple FPT branching algorithm in time $\mathcal{O}^*(2^k)$.

As observed by Cai and Juedes [55], this idea applies also to problems in planar graphs. For instance, FPT algorithms with running time $\mathcal{O}^*(2^{o(\sqrt{k})})$ for problems like Planar Vertex Cover, Planar Independent Set, or Planar Dominating Set would have also classical complexity

$2^{o(\sqrt{N})}$ where N is the number of vertices of the graph. We have however already refuted existence of such algorithms under ETH. Note moreover that these lower bounds match the upper bounds given by the bidimensionality theory (we discussed bidimensionality in Section 3.2.4): all three aforementioned problems are in fact solvable in time $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k})})$. Therefore, the square root in the exponent of algorithms obtained using bidimensionality is not just an artifact of the technique, but an intrinsic property of the problems' complexity. Understanding this 'square root phenomenon' is one of the main goals of the bidimensionality theory, to which the optimality program clearly contributes.

We remark that in order to exclude existence of an $\mathcal{O}^*(2^{o(k)})$ algorithm under ETH it is sufficient to provide a polynomial-time reduction from 3-CNF-SAT that takes a formula with n variables and m clauses, and produces an equivalent instance of the target problem whose size may be superlinear, but whose parameter is linear in $n+m$. Pipelining such a reduction with a hypothetical subexponential algorithm for the problem would yield a $2^{o(n+m)}$ algorithm for 3-CNF-SAT, thus contradicting ETH by Corollary 5. More generally, to exclude running time $\mathcal{O}^*(2^{o(f(k))})$ the output instance should have parameter bounded by $\mathcal{O}(f^{-1}(n+m))$, where f^{-1} is the inverse function of f; see also [132]. All the more recent lower bounds under ETH, including the ones given in Chapters 8 and 9 of this thesis, in fact use this framework; that is, one takes care of the output parameter only and ignores the intermediate step of considering the total size of the output instance.

4.2 The mysterious class SUBEPT

Recall that class SUBEPT, defined by Flum and Grohe [132, Chapter 16], consists of parameterized problems solvable in $\mathcal{O}^*(2^{o(k)})$ time where k is the parameter. While the bidimensionality theory is a natural explanation of existence of subexponential algorithms for problems on geometric graph classes, there exist also other families of problems that belong to SUBEPT.

The first natural examples of problems belonging to SUBEPT outside the framework of bidimensionality were found in the context of **tournaments**, with the most important example of the Feedback Arc Set problem [10, 126, 205]. In Part II we consider parameterized problems in tournaments, and in particular we design a few new subexponential parameterized algorithms. We refer to the introductory chapter of this part for an overview of the topic.

Recently, some surprising subexponential algorithms were found for **edge modification problems**. In these problems one is given a graph G and an integer k, which is the parameter, and the goal is to add or remove (depending on the variant) at most k edges to make G satisfy some property. *Deletion* problems allow only edge removals, *completion* problems allow only edge additions, while *edition* problems allow both operations. The first breakthrough result is due to Fomin and Villanger [149] who have shown that the Minimum Fill-in problem, i.e., edge completion to a chordal graph (a graph with no induced cycle longer than 3), can be solved in time $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k}\log k)})$, thus substantially improving upon a number of previous algorithms with running times $\mathcal{O}^*(2^{\mathcal{O}(k)})$. Later, Ghosh et al. [165] have proven that also edge completion and edge deletion to a split graph (a graph whose vertex set can be partitioned into a clique and an independent set) can be solved in $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k}\log k)})$ time.

Contrary to the results given by the bidimensionality theory, we do not have matching lower bounds for the aforementioned two families of problems. As Fomin and Villanger [149] observe, the known NP-hardness reduction for Minimum Fill-in excludes only existence of an algorithm with running time $\mathcal{O}^*(2^{o(k^{1/6})})$ under ETH, so we still have a large gap between the upper and the lower bound. Similarly weak lower bounds follow from the NP-hardness reductions for the Feedback

Arc Set in Tournament problem of Alon [9] and of Charbit et al. [60]. On the other hand, the subexponential algorithms for problems in tournaments and for edge modification problems can be obtained via a number of very different techniques, like potential maximal cliques [149], chromatic coding [10], or the technique of k-cuts that is presented in this thesis. For each of these techniques the running time $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k}\cdot\mathrm{polylog}(k))})$ seems very natural. Finding matching lower bounds for all these problems is arguably one of the most interesting open problems in the optimality program.

In Chapter 8 we make a step towards this goal. We consider there the Cluster Editing problem: add or remove at most k edges of a given graph to obtain a cluster graph, i.e., a disjoint union of cliques (called *clusters*). It appears that the problem in general does not admit a subexponential parameterized algorithm; however, if one assumes that the target number of clusters is p, then an algorithm with running time $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{pk})})$ can be designed. Thus, the problem can be solved in subexponential parameterized time whenever p is a sublinear function of k. Moreover, we have obtained also a matching lower bound under ETH for all magnitudes of $p = p(k)$ between a constant and a linear function of k (see Chapter 8 for a precise statement of this claim). As a result, Cluster Editing is perhaps the first edge modification problems for which such a thorough complexity analysis has been performed.

4.3 Slightly superexponential parameterized time

As we have seen, for many problems already the known NP-hardness reductions are sufficient for settling tight bounds on their parameterized complexity. Typically such problems admit an FPT with simple single exponential dependence on the parameter, which is matched by a classical reduction from 3-CNF-SAT that produces an instance with parameter bounded linearly in the size of the input formula. For some important classes of problems, however, the known FPT algorithms have higher running times, which is therefore not matched by lower bounds derived in this manner.

The first such class to be studied systematically was the class of problems solvable in $\mathcal{O}^*(2^{\mathcal{O}(k \log k)})$ time, also called *slightly superexponential*; this study is due to Lokshtanov et al. [241]. Such running time appears naturally for branching FPT algorithms where the depth of the search tree is bounded by $\mathcal{O}(k)$, but at each branching step the algorithm chooses one of $k^{\mathcal{O}(1)}$ possibilities. Lokshtanov et al. start with identifying problems that one could call canonical for this class; in other words, problems where one is asked to make k independent 1-in-k choices that can be checked against each other. The first problem defined by Lokshtanov et al. is $k \times k$ Clique: given a graph G on vertex set $\{1, 2, \ldots, k\} \times \{1, 2, \ldots, k\}$, interpreted as a $k \times k$ matrix, check if there exists a clique in G that contains exactly one vertex from each row. The second problem is $k \times k$ Hitting Set: given a family $\mathcal{F}$ of subsets of $\{1, 2, \ldots, k\} \times \{1, 2, \ldots, k\}$, again interpreted as a $k \times k$ matrix, verify whether there is a set $X \subseteq \{1, 2, \ldots, k\} \times \{1, 2, \ldots, k\}$ that contains exactly one element from each row and that has nonempty intersection with each element of $\mathcal{F}$.

Observe that both $k \times k$ Clique and $k \times k$ Hitting Set can be solved in $\mathcal{O}^*(k^k)$ time by checking all possible choices of one vertex from each row of the matrix. Lokshtanov et al. prove that a substantial improvement of these brute-force algorithms is unlikely: existence of an algorithm for any of these problems with running time $\mathcal{O}^*(2^{o(k \log k)})$ would contradict ETH. The idea of the proof is to first show a lower bound for $k \times k$ Clique, and then reduce it further to $k \times k$ Hitting Set via a sequence of intermediate steps.

The crucial step is of course the lower bound for $k \times k$-Clique. Here, one essentially needs to find a reduction from any problem known not to admit a subexponential algorithm that produces

an instance of $k \times k$-Clique with $k = \mathcal{O}(n/\log n)$, where n is the size of the input instance. Hence, in some sense we need to devise a way to 'repack' the information from a 2^n search space in the input instance to a k^k search space in the output instance. Lokshtanov et al. find it most convenient to start with an instance of 3-Coloring, and divide vertices of the input graph into around $\Theta(n/\log n)$ groups of size around $\Theta(\log n)$ each. Intuitively, choosing a coloring of each group can thus express one of $3^{\Theta(\log n)} = n^{\Theta(1)}$ choices. Then groups correspond to rows of the matrix, while colorings of one group correspond to vertices of the corresponding row.

Having established the canonical problems as a base for reductions, Lokshtanov et al. provide a number of corollaries. For instance, they consider the Closest String problem defined as follows: given a finite alphabet Σ, two integers d, ℓ and a family of strings $u_1, u_2, \ldots, u_t$ over Σ of length ℓ each, decide whether there exists a string $u \in \Sigma^\ell$ that is in Hamming distance at most d from each of u_i, i.e., for each $i = 1, 2, \ldots, t$ it holds that u and u_i differ on at most d letters. The Closest String problem can be solved in $\mathcal{O}^*(2^{\mathcal{O}(d \log d)})$ time [171] and in $\mathcal{O}^*(2^{\mathcal{O}(d \log |\Sigma|)})$ time [246]. Using the prepared tools Lokshtanov et al. were able to prove that both these upper bounds are essentially tight: both existence of an algorithm running in $\mathcal{O}^*(2^{o(d \log d)})$ time or an algorithm running in $\mathcal{O}^*(2^{o(d \log |\Sigma|)})$ time would contradict ETH. Apart from this, Lokshtanov et al. proved also a similar lower bound for the Distortion problem that treats of embedding a graph into a metric space while approximately preserving distances, and for the Vertex Disjoint Paths problem parameterized by the treewidth of the input graph. The latter result will be discussed in more details in the next section.

4.4 Dynamic programming on treewidth

It has been observed by Lokshtanov et al. [239] that SETH can be used conveniently for proving lower bounds on the running time of dynamic programming routines on tree decompositions. Recall that many classical NP-hard problems admit efficient FPT algorithms when parameterized by treewidth obtained via the dynamic programming principle. Existence of such algorithms can be derived from expressibility in $\mathbf{MSO}_2$ by the results of Courcelle (Theorem 16) and of Arnborg et al. [16]; however, in many cases one can design an explicit dynamic program which works in simple single exponential time, or slightly superexponential time. For instance, for the Independent Set and Dominating Set problems the running times on a tree decomposition of width t are $\mathcal{O}(2^t \cdot t^{\mathcal{O}(1)} \cdot n)$ and $\mathcal{O}(3^t \cdot t^{\mathcal{O}(1)} \cdot n)$, respectively.

The question asked by Lokshtanov et al. [239] was whether the bases of the exponents (i.e., 2 and 3 in these examples) can be substantially improved. As they show, this is implausible under SETH. More precisely, unless the general CNF-SAT can be solved in $(2-\varepsilon_0)^n$ time for some $\varepsilon_0 > 0$ (note that this assumption is slightly weaker than SETH), no algorithms for Independent Set or Dominating Set can achieve running times $\mathcal{O}^*((2-\varepsilon)^p)$ and $\mathcal{O}^*((3-\varepsilon)^p)$, respectively, where p is the width of a given path decomposition and $\varepsilon > 0$ is any positive constant.

Let us shortly discuss the methodology used for obtaining such results. We provide a reduction from the CNF-SAT problem; assume then that we are given a CNF formula φ with n variables and m clauses. The reduction should in polynomial time construct a graph G with a path decomposition of width roughly n and compute an integer k, such that G admits an independent set of size k if and only if φ is satisfiable. The idea is to examine closely the known dynamic programming routine for the Independent Set problem. Intuitively, the states of the dynamic program correspond to partitions of the bag into vertices that are included and excluded from the constructed independent

set; thus, we have at most 2^{t+1} states per bag. Now the goal is to construct G in such a manner that each of these 2^{t+1} states corresponds to one of 2^n possible valuations of variables of φ. Graph G is constructed by taking a sequence of $\Theta(m)$ bags. In each bag we test satisfaction of one clause by attaching a constant-treewidth verificator gadget, and moreover for each two consecutive bags we need to ensure that the choice of the state in the first one is the same as in the second one[2]. Thus, by choosing the intersection of the constructed independent set with the first bag we in fact choose an evaluation of variables of φ, and further bags verify whether this evaluation satisfies φ.

For Dominating Set the construction must be more involved, as we need to pack search space of size 2^n of a CNF-SAT instance into 3^{t+1} states of the dynamic program. Thus, the width of the constructed path decomposition should be roughly $n/\log 3$ and one needs to devise a clever way of repacking information. Lokshtanov et al. have extended the technique also to other problems that were known to admit simple single exponential algorithms parameterized by treewidth: for instance, for q-Coloring the lower bound is $\mathcal{O}^*((q-\varepsilon)^p)$ and for Odd Cycle Transversal the lower bound is $\mathcal{O}^*((3-\varepsilon)^p)$, for any $\varepsilon > 0$.

As mentioned in the previous section, in the second paper [241] Lokshtanov et al. proved also a lower bound on the running time of the dynamic programming on a tree decomposition for Vertex Disjoint Paths. More precisely, no algorithm for Vertex Disjoint Paths can achieve running time $\mathcal{O}^*(2^{o(p \log p)})$ unless ETH fails, where p is the width of a given path decomposition[3]. The idea is that the classical dynamic program for Vertex Disjoint Paths has $2^{\Theta(t \log t)}$ states per bag, and one needs to translate the k^k search space of the $k \times k$ Hitting Set problem to $2^{\Theta(t \log t)}$ states of the dynamic program, in a similar way as the 2^n search space of CNF-SAT was translated to $2^{\Theta(t)}$ states of simple single exponential dynamic programs.

An immediate follow-up question of the works of Lokshtanov et al. [239, 241] was whether other natural dynamic programming routines working in time $\mathcal{O}^*(2^{\mathcal{O}(t \log t)})$ are optimal as well. Of particular interest was a number of problems with connectivity constraints, like Hamiltonian Path, Connected Vertex Cover, or Connected Dominating Set [239]; for these problems, the natural family of states is essentially formed by partitions of the bag, whose number is $2^{\Theta(t \log t)}$ in the worst case. This has been later disproved by Cygan et al. [91] (see also the full version [92]), who have presented a technique dubbed *Cut&Count* using which one can obtain randomized simple single exponential algorithms for many connectivity problems (see also [268] for a unified formalism via a form of modal logics). For instance, it has been shown in [91] that Hamiltonian Path can be solved in $\mathcal{O}^*(4^t)$ time, Connected Vertex Cover in $\mathcal{O}^*(3^t)$ time and Connected Dominating Set in $\mathcal{O}^*(4^t)$ time, where t is the width of a given tree decomposition.

Cygan et al. augmented their results with a variety of lower bounds in the spirit of the earlier work of Lokshtanov et al. [239, 241]. In particular, for many problems it turned out that the upper bound given by Cut&Count can be matched by a lower bound: for example, Connected Vertex Cover cannot be solved in $\mathcal{O}^*((3-\varepsilon)^p)$ time and Connected Dominating Set in $\mathcal{O}^*((4-\varepsilon)^p)$ time unless SETH fails, where p is the width of a given path decomposition and $\varepsilon > 0$ is any positive constant. This proves that the obtained running times are not just artifacts of the technique, but rather inherent properties of the corresponding problems. Moreover, Cygan et al. have also shown a number of slightly superexponential lower bounds, thus sketching the limits of the technique. For example, the question whether *at most* r vertex-disjoint cycles can be found in a graph can

[2] This is not entirely true in the technical construction of [239], but the intuition is preserved.

[3] The result is stated in [241] only for treewidth, but a careful examination of the proof shows that it holds also for pathwidth.

be solved in $\mathcal{O}^*(4^t)$ time, while after replacing *at most* by *exactly* or *at least* a $\mathcal{O}^*(2^{o(p \log p)})$ lower bound under ETH can be given.

It is worth mentioning that the motivation of the work of Cygan et al. [91] was actually proving slightly superexponential lower bounds for the aforementioned connectivity problems, and the Cut&Count technique was discovered only after numerous attempts of constructing reductions. This is a clear example of an achievement obtained by a systematic study of the optimality program.

Later on, Bodlaender et al. [43] have derandomized the Cut&Count technique at a cost of larger bases of the exponents using a new rank-based approach, and also made the polynomial factors of the algorithms linear. The particular case of Hamiltonian Cycle was considered by Cygan et al. [88], since the upper bound given in [91] was not matched by a lower bound. Surprisingly, it turned out that Hamiltonian Cycle can be solved in $\mathcal{O}^*((2+\sqrt{2})^p)$ randomized time, where p is the width of a given path decomposition, while an $\mathcal{O}^*((2+\sqrt{2}-\varepsilon)^p)$ algorithm for any $\varepsilon > 0$ would give a $\mathcal{O}((2-\varepsilon')^n)$ algorithm for CNF-SAT, thus contradicting SETH. This tight bound on the base of the exponent being $2+\sqrt{2}$ is certainly one of the biggest surprises of the optimality program. Unfortunately, so far it is not known whether also an algorithm working in $\mathcal{O}^*((2+\sqrt{2})^t)$ time on a tree decomposition of width t can be designed. The obstacle is the join step: it is unclear whether with the current formulation of the dynamic program one can avoid considering all pairs of states via any kind of a fast convolution.

4.5 Lower bounds for XP algorithms

Let us consider the Clique problem. While a trivial algorithm with running time $n^{\mathcal{O}(k)}$ exists, we do not expect the problem to be solvable in fixed-parameter tractable time, for existence of such an algorithm would imply that $FPT = W[1]$. However, these lower and upper bounds still have much room for improvement: for instance, would it be possible that Clique solvable in $n^{\mathcal{O}(\sqrt{k})}$ time? Or maybe even $n^{\mathcal{O}(\log \log k)}$ time would be possible? The sole assumption of $FPT \neq W[1]$ seems not strong enough to give a satisfactory answer to this question; however, it turns out that we may provide much tighter lower bounds assuming ETH.

The question of optimality of XP algorithms for $W[1]$- and $W[2]$-hard problem was first considered by Chen et al. [64, 66, 67]. The take-away message of this study is that assuming nonexistence of a subexponential algorithm for 3-CNF-SAT, no algorithm for Clique, Independent Set, Dominating Set, Set Cover, and many other $W[1]$- and $W[2]$-hard problems, can achieve running time $f(k) \cdot n^{o(k)}$ for any computable function f. Note that this means that ETH implies that $FPT \neq W[1]$. We remark that for problems that are $W[2]$-hard, like Dominating Set or Set Cover, even a weaker assumption of $FPT \neq W[1]$ suffices to state the same lower bounds [67].

The parameterized complexity of Dominating Set has been later revisited by Pătraşcu and Williams [272], who provided a sharper lower bound assuming SETH. They proved that while Dominating Set can be solved in $n^{k+o(1)}$ time for any $k \geq 7$, existence of an algorithm solving Dominating Set in time $\mathcal{O}(n^{k-\varepsilon})$ for any fixed $k \geq 3$ and $\varepsilon > 0$ would imply an algorithm for the general CNF-SAT problem with running time $(2-\varepsilon')^n$ for some $\varepsilon' > 0$.

The results of Chen et al. [64, 66, 67] provide a convenient base of further reductions, using which one can prove lower bounds on running times of XP algorithms for other problems not likely to be in FPT. For instance, in order to show that a parameterized problem L does not admit an algorithm with running time $f(k) \cdot n^{o(k)}$ for any computable function f, it suffices to present a reduction from the Clique problem to L with the following properties. First, the reduction works

in $g(k) \cdot n^{\mathcal{O}(1)}$ time for some computable function g, where k is the parameter of the input instance of Clique. Second, it produces an equivalent instance of L of size polynomial in the input size and with parameter bounded linearly in k. Such reductions are called by Chen et al. *linear FPT reductions* (see [66] for a more detailed definition). As they observe, many of $W[1]$-hardness and $W[2]$-hardness reductions present in the literature are in fact linear.

More generally, to exclude running time $f(k) \cdot n^{o(m(k))}$ for any computable function f, the reduction should produce an output instance with parameter bounded by $\mathcal{O}(m^{-1}(k))$, where m^{-1} is the inverse function of m. The framework of such parameter-preserving reductions has been successfully applied several times to give tight bounds on the complexity of various XP algorithms; let us now survey a few examples.

Fomin et al. [133] considered problems parameterized by cliquewidth, and provided matching upper and lower bounds for two of them: Max-Cut and Edge Dominating Set (we refer to [133] for problems' definitions). More precisely, both problems can be solved[4] in $n^{\mathcal{O}(k)}$ time on graphs of cliquewidth k, while existence of an algorithm for any of them achieving running time $f(k) \cdot n^{o(k)}$ for any computable function f would contradict ETH. We remark that the study of Fomin et al. was a continuation of an earlier work [134], whose main achievement was settling $W[1]$-hardness of cliquewidth parameterizations for a number of classical problems. For many problems considered in [134] we still lack matching lower and upper bounds, with a notable example of Hamiltonian Path ($n^{\mathcal{O}(k^2)}$ upper bound, $n^{o(k)}$ lower bound).

Similarly as in the case of bidimensionality, also in this setting ETH can be also used to justify appearance of a square root for problems with geometric constraints. For instance, Marx [248] has given a reduction from Clique to Independent Set on unit disk graphs that produces an instance with parameter bounded quadratically in the input parameter. Thus, as an immediate corollary we obtain that Independent Set on unit disk graphs does not admit an algorithm with running time $f(k) \cdot n^{o(\sqrt{k})}$ for any computable function f, unless ETH fails. This lower bound can be matched by an upper bound of $n^{\mathcal{O}(\sqrt{k})}$ [8]. Interestingly, the main motivation of the work of Marx was proving lower bounds on running times of approximation schemes of classical problems on geometric graphs; this shows applicability of the optimality program beyond parameterized complexity.

Later, Klein and Marx [217, 251] have provided similar matching bounds for the Edge Multiway Cut problem on planar graphs. The problem is as follows: given a graph G with a set of terminals $T \subseteq V(G)$ and an integer k, one is asked to remove at most k edges of the graph so that no two terminals remain in the same connected component. While the problem is NP-hard on general graphs already for $|T| = 3$ [97], an XP algorithm with running time $n^{\mathcal{O}(|T|)}$ was known for planar graphs [187]. It had been then a long-standing open problem whether this result can be improved to FPT time [119]. This question has been resolved by Marx [251], who has shown that this is probably not the case: the problem is $W[1]$-hard. In a companion paper together with Klein [217] they have also shown an XP algorithm essentially matching the lower bound implied by [251]. More precisely, Edge Multiway Cut can be solved in time $2^{\mathcal{O}(|T|)} \cdot n^{\mathcal{O}(\sqrt{|T|})}$ [217], while existence of an algorithm achieving running time $f(|T|) \cdot n^{o(\sqrt{|T|})}$ for any computable function f would contradict ETH [251].

Marx [250] has also considered optimality of treewidth-based algorithms for solving *constraints-satisfaction problems* (CSPs). It is known that a CSP can be solved by a standard dynamic program in time $n^{\mathcal{O}(t)}$, where n is the total input size (size of the domain plus size of the primal graph) and t is the treewidth of the primal graph; see [250] for appropriate definitions. Marx has shown that this result cannot be substantially improved: assuming ETH, there is no recursively enumerable class of

[4] We assume that the algorithm is given an appropriate clique expression constructing the graph.

graphs $\mathcal{G}$ with unbounded treewidth, for which there exists an algorithm solving CSPs with primal graphs from $\mathcal{G}$ in time $f(G) \cdot n^{o(t/\log t)}$, where $f(G)$ is any function of the input graph. Marx also provides a number of strong corollaries of this result, for instance in the context of the Subgraph Isomorphism problem.

Finally, probably the most surprising tight result has been given by Marx [249] for the Closest Substring problem, a close relative of the Closest String problem considered in Section 4.3. In this problem one is given a finite alphabet Σ, a sequence of strings $u_1, u_2, \ldots, u_k$ over Σ, and integers L and d. The question is to find one string u of length L, such that each u_i contains a consecutive substring u_i^* of length L that is in Hamming distance at most d from u, i.e., u and u_i^* differ on at most d letters. First, Marx shows two non-trivial algorithms solving the problem in times $f(d) \cdot n^{\mathcal{O}(\log d)}$ and $g(k,d) \cdot n^{\mathcal{O}(\log\log k)}$, respectively, for some computable functions f, g. The surprise is that both of these running times can be shown to be essentially tight. More precisely, assuming ETH there is no algorithm solving Closest Substring in time $f'(k,d) \cdot n^{o(\log d)}$ or in time $g'(k,d) \cdot n^{o(\log\log k)}$ for any computable functions f', g'. In order to exclude such running times, Marx needed to construct a reduction from the Clique problem with parameter t that produces an instance of Closest Substring with $d = 2^{\mathcal{O}(t)}$ and $k = 2^{2^{\mathcal{O}(t)}}$.

4.6 Linking brute-force and dynamic programming to SETH

We conclude by reviewing the results of Cygan et al. [86] on optimality of brute-force and dynamic programming algorithms under SETH. The starting point of the study of Cygan et al. is an observation that many classical parameterized problems solvable in $\mathcal{O}^*(2^k)$ time, where k is the parameter, can be classified into two categories.

- *Search problems* that can be solved in $\mathcal{O}^*(2^k)$ time by a brute-force check of a search space of size 2^k. Examples of such problems are q-CNF-SAT for q tending to infinity, Set Cover parameterized by the size of the family, or Set Splitting (see [86] for the missing problem definitions).
- *Covering problems* that can be solved in $\mathcal{O}^*(2^k)$ time by dynamic programming on subsets of a universe of size k. Examples of such problems are Set Cover parameterized by the size of the universe, Set Partitioning or Steiner Tree.

The goal of Cygan et al. was to show that breaking the 2^k barrier for all these problems is equivalent, that is, existence of an algorithm for one of them that runs in time $\mathcal{O}^*((2-\varepsilon)^k)$ for some $\varepsilon > 0$ implies existence of such algorithms for all of them, and in particular it refutes SETH. This goal has been achieved only partially. Cygan et al. managed to show equivalence of all the considered search problems and that non-existence of better algorithms for the covering problems can be reduced to one canonical case of Set Cover parameterized by the size of the universe. However, the last reduction eluded the authors, so it may still be that Set Cover parameterized by the universe size can be solved in $\mathcal{O}^*((2-\varepsilon)^n)$ time for some $\varepsilon > 0$ while SETH holds. Cygan et al. expect that the missing link exists, and therefore state a conjecture that Set Cover cannot be solved in $\mathcal{O}^*((2-\varepsilon)^n)$ time for any $\varepsilon > 0$, where n is the cardinality of the universe.

To justify this claim, Cygan et al. observe that with high probability an attempt of breaking the 2^n barrier for Set Cover using standard randomized-algebraic tools would also give an algorithm that counts the number of set covers modulo 2 in $\mathcal{O}^*((2-\varepsilon)^n)$ time. However, when one considers counting solutions modulo 2 instead of just existence of solutions, then the missing link can be

found! More precisely, the authors show that if for some $\varepsilon > 0$ and every q there exists an algorithm counting modulo 2 the number of set covers in $\mathcal{O}^*((2-\varepsilon)^n)$ time in instances where the sets in the family are of size at most q, then for some $\varepsilon' > 0$ and every q there exists an algorithm counting modulo 2 the number of satisfying assignments of a q-CNF formula in time $\mathcal{O}^*((2-\varepsilon')^n)$, where n is the number of variables. Existence of such an algorithm, however, is known to violate SETH via standard isolation techniques [56, 304]. Therefore, assuming SETH any algorithm for Set Cover with running time $\mathcal{O}^*((2-\varepsilon)^n)$ for some $\varepsilon > 0$ cannot be able to count the number of solutions modulo 2, which means that it would probably need to be based on a new approach, still to be discovered.

Part II

Topological problems in semi-complete digraphs

Chapter 5

Introduction to tournaments and semi-complete digraphs

5.1 Introduction

5.1.1 The theory of graph minors for digraphs

As we argued in Section 3.1.5, it is a very natural and important question, whether techniques and results of Graph Minors can be applied in the world of *directed* graphs or digraphs. In spite of many attempts, we are still very far from the right answer. Even to capture a good analogue for treewidth in digraphs is a non-trivial task and several notions like directed treewidth [198], DAG-width [30] or Kelly-width [194] can be found in the literature; recall that we have surveyed these propositions in Section 3.4. However, none of them shares all the "nice" properties of undirected treewidth. In fact this claim can be formalized and proved; Ganian et al. [159] argued that "*any reasonable algorithmically useful and structurally nice digraph measure cannot be substantially different from the treewidth of the underlying undirected graph*".

The notion of a graph minor is crucial in defining obstructions to small treewidth in an undirected graph. There are several ways to generalize this definition to digraphs and, as in case of treewidth, it is unclear which of them is the most natural. One approach is to consider topological embeddings or immersions. An undirected graph H is a *topological subgraph* (or *topological minor*) of an undirected graph G if a subdivision of H is a subgraph of G. In other words, graph H can be embedded into graph G in such a way that vertices of H are mapped to pairwise different vertices of G, edges of H are mapped to vertex-disjoint paths in G. An *immersion* of a graph H into a graph G is defined like a topological embedding, except that edges of H correspond to edge-disjoint paths in G. Both these notions can be naturally translated to directed graphs by replacing paths with directed paths.

It were long-standing open questions whether deciding if an undirected graph H can be topologically embedded (immersed) into G is fixed-parameter tractable, when parameterized by the size of H. Both questions were answered positively only very recently by Grohe, Kawarabayashi, Marx, and Wollan [174]. Unfortunately, the work of Grohe et al. cannot be extended to directed graphs. By the classical result of Fortune, Hopcroft, and Wyllie [151] the problem of testing whether a given digraph G contains H as a (directed) topological subgraph is NP-complete even for very simple digraphs H of constant size. Similar results can be easily obtained for immersions. In

fact, what Fortune et al. [151] showed is that the Vertex (Edge) Disjoint Paths problems are NP-complete on general digraphs even for $k = 2$, and the hardness of topological containment and immersion testing are simple corollaries of this fact. Therefore, Vertex (Edge) Disjoint Paths were studied intensively on different classes of directed graphs. For example if we constrain the input digraphs to be acyclic, then both variants still remain NP-complete when k is part of the input [125], but are polynomial-time solvable for every constant number of terminal pairs k [151], which was not the case in the general setting [151]. Slivkins [300] has shown that both problems are in fact $W[1]$-hard when parameterized by k, thus completing the picture of their parameterized complexity in this restricted case.

5.1.2 The containment theory for tournaments

Tournaments form an interesting and mathematically rich subclass of digraphs. Formally, a simple digraph T is a tournament if for every pair of vertices v, w, **exactly** one of arcs (v, w) or (w, v) is present in T. We also consider a superclass of tournaments, called semi-complete digraphs, where we require **at least** one of arcs (v, w) or (w, v) to be present in T, thus allowing both of them to be present at the same time. Many algorithmic problems were studied on tournaments, with notable examples of problems strongly related to this work: Vertex (Edge) Disjoint Paths and Feedback Arc (Vertex) Set problems. We refer to the book of Bang-Jensen and Gutin [22] for a more thorough introduction to algorithms for digraphs, and in particular for tournaments.

The work on topological problems in semi-complete digraphs began perhaps with the work of Bang-Jensen and Thomassen [21, 23], who showed that in spite of the fact that the Vertex (Edge) Disjoint Paths problems remain NP-complete on tournaments when k is a part of the input, they are solvable in polynomial time for the case $k = 2$ even on semi-complete digraphs. This line of research was later continued by Chudnovsky, Fradkin, Kim, Scott, and Seymour [72, 73, 74, 152, 153, 216] (CFKSS, for short).

The recent work of CFKSS drastically advanced the study of minor-related problems in semi-complete digraphs by building an elegant containment theory for this class. The central notions of the theory are two width measures of digraphs: *cutwidth* and *pathwidth*. The first one is based on vertex orderings and resembles classical cutwidth in the undirected setting [302], with the exception that only arcs directed forward in the ordering contribute to the cut function. The second one is a similar generalization of undirected pathwidth; recall that we have surveyed the work on directed pathwidth in general digraphs in Section 3.4.3. For the sake of introduction we do not need formal definitions of cutwidth and pathwidth that are somewhat technical; however, the reader may find them in Section 5.2.3.

Chudnovsky, Fradkin, and Seymour [72] proved a structural theorem that provides a set of obstacles for admitting an ordering of small (cut)width; a similar theorem for pathwidth was proven by Fradkin and Seymour [153]. A large enough obstacle for cutwidth admits every fixed-size digraph as an immersion, and the corresponding is true also for pathwidth and topological containment. Basing on the first result, Chudnovsky and Seymour [74] were able to show that immersion is a well-quasi-ordering on the class of semi-complete digraphs. Indeed, following the same line of reasoning as in the Graph Minors project (see Section 3.1.1), it is sufficient to prove the claim for the class of semi-complete digraphs that exclude some fixed semi-complete digraph H_0 as an immersion. Using the structural theorem we infer that such digraphs have cutwidth bounded by a constant. For graphs of constant cutwidth, however, the well-quasi-ordering claim can be proven using a more direct approach via Higman's lemma.

Unfortunately, the same reasoning breaks for topological containment, since it is already not true that topological containment is a well-quasi-ordering of graphs of constant pathwidth. However, Kim and Seymour [216] have recently introduced a slightly different notion of a *minor* order, which indeed is a well-quasi-ordering of semi-complete digraphs.

As far as the algorithmic aspects of the work of CFKSS are concerned, the original proofs of the structural theorems can be turned into approximation algorithms, which given a semi-complete digraph T and an integer k find a decomposition of T of width $\mathcal{O}(k^2)$, or provide an obstacle for admitting decomposition of width at most k. For cutwidth the running time on a semi-complete digraph on n vertices is $\mathcal{O}(n^3)$, but for pathwidth it is $\mathcal{O}(n^{\mathcal{O}(k)})$ for some function m; this excludes usage of this approximation as a subroutine in any FPT algorithm, e.g. for topological containment testing.

As Chudnovsky, Fradkin, and Seymour [72] observe, existence of an FPT approximation algorithm for cutwidth allows us to design FPT algorithms for checking whether a given digraph H can be immersed into a semi-complete digraph T. Consider the following WIN/WIN approach. We run the approximation algorithm for cutwidth for some parameter that is a (large) function of $|H|$. In case a decomposition of width bounded by a function of $|H|$ is returned, we can employ a dynamic programming routine on this decomposition that solves the problem in FPT time. Otherwise, the approximation algorithm provided us with a large combinatorial obstacle into which every digraph of size at most $|H|$ can be embedded. Therefore, we can safely provide a positive answer. Fradkin and Seymour [153] observe that the same approach can be applied to topological subgraph testing using their approximation algorithm for pathwidth instead. However, this approximation algorithm does not work in FPT time, so the obtained topological containment test is also not fixed-parameter tractable. Let us remark that the original dynamic programming routine for topological containment working on a path decomposition, presented by Fradkin and Seymour [153], was also not fixed-parameter tractable.

The approximation algorithms for cutwidth and for pathwidth either provide an obstacle into which every digraph of size at most ℓ can be embedded, or construct a decomposition of width $f(\ell)$ for some multiple-exponential function f (yet elementary). Therefore, the obtained algorithm for immersion testing also inherits this multiple-exponential dependence on the size of the digraph to be embedded, i.e., it works in time $f(|H|) \cdot n^3$ for some function f that is multiple-exponential, yet elementary. For topological containment this problem is even more serious, since we obtain multiple-exponential dependence on $|H|$ in the exponent of the polynomial factor, and not just in the multiplicative constant standing in front of it.

One of the motivations of the work of CFKSS was extending the work of Bang-Jensen and Thomassen on the Vertex (Edge) Disjoint Paths problems. The new containment theory turned out to be capable of answering many questions, yet not all of them. Using the approximation algorithm for cutwidth, Fradkin and Seymour [152] designed an FPT algorithm for Edge Disjoint Paths working in time $f(k) \cdot n^5$ for some function f that is multiple-exponential, yet elementary. The algorithm uses again the WIN/WIN approach: having approximated cutwidth, we can either find an ordering of small width on which a dynamic program can be employed, or we find a large combinatorial obstacle. In this case, Chudnovsky, Fradkin, and Seymour are able to identify an irrelevant vertex in the obstacle, which can be safely removed without changing existence of a solution. That is, they design an irrelevant vertex rule. For the Vertex Disjoint Paths problem, Chudnovsky, Scott, and Seymour [73] give an XP algorithm using a different approach. To the best of author's knowledge, the question whether the Vertex Disjoint Paths problem admits an FPT algorithm in semi-complete digraphs, is still open.

The Edge Disjoint Paths problem is a special case of the Rooted Immersion problem defined as follows: given a digraph H with prescribed pairwise different vertices $u_1, u_2, \ldots, u_h$, called *roots*, and a semi-complete digraph T also with pairwise different roots $v_1, v_2, \ldots, v_h$, we ask whether there exists an immersion of H in T that *preserves roots*, that is, maps each u_i to corresponding v_i. The Edge Disjoint Paths problem can be hence modeled as follows: we take H to be a digraph consisting of k independent arcs, and all the vertices of H are roots required to be mapped to respective endpoints of the paths that we seek for. In the same manner we may define Rooted Topological Containment problem that generalizes the Vertex Disjoint Paths problem. It appears that the FPT algorithm for Edge Disjoint Paths of Fradkin and Seymour can be generalized to solve also the Rooted Infusion problem, which is a relaxation of Rooted Immersion where we do not require the images of vertices of H to be distinct. Note that Rooted Infusion also generalizes Edge Disjoint Paths in the same manner as Rooted Immersion does. As Fradkin and Seymour admit, they were not able to solve the Rooted Immersion problem in FPT time using their approach.

As far as computation of cutwidth and pathwidth exactly is concerned, by the well-quasi-ordering result of Chudnovsky and Seymour [74] we have that the class of semi-complete digraphs of cutwidth bounded by a constant is characterized by a finite set of forbidden immersions; the result of Kim and Seymour [216] proves that we can infer the same conclusion about pathwidth and minors. Having approximated the corresponding parameter in FPT or XP time, we can check if any of these forbidden structures is contained in a given semi-complete digraph using dynamic programming. This gives FPT and XP exact algorithms for computing cutwidth and pathwidth, respectively; however, they are both non-uniform — the algorithms depend on the set of forbidden structures which is unknown — and non-constructive — they provide just the value of the width measure, and not the optimal decomposition. Also, we have virtually no control on the dependence of the running time on the target width value.

5.1.3 Tournaments and parameterized complexity

The class of tournaments received also a lot of attention from the point of view of parameterized complexity, perhaps because of a vast number of positive results that can be obtained in this setting. The two problems that have been most intensively studied are Feedback Arc Set and Feedback Vertex Set, defined as follows: given a tournament T and an integer k, we ask whether one can delete at most k arcs (vertices) to obtained an acyclic digraph.

Feedback Arc Set in Tournaments (FAST, for short) was perhaps the first natural parameterized problem outside the framework of bidimensionality shown to admit a subexponential parameterized algorithm. The first such algorithm, with running time $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k}\log k)})$, is due to Alon, Lokshtanov, and Saurabh [10]. This has been further improved by Feige [126] and by Karpinski and Schudy [205], who have independently shown two different algorithms with running time $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k})})$. The work of Alon et al. introduced a novel technique called *chromatic coding*, while the algorithms of Feige and of Karpinski and Schudy were based on the degree ordering approach, and the techniques developed there were more contrived to the problem. These results led to large interest in parameterized problems on tournament-like and generally dense settings, as it turned out that the technique of chromatic coding proves to be useful also in other problems [130, 139, 165, 181].

From the point of view of kernelization, FAST admits a kernel with at most $(2+\epsilon)k$ vertices for any $\epsilon > 0$ [32]. The Feedback Vertex Set in Tournaments problem is known to admit a kernel with at most $\mathcal{O}(k^3)$ vertices and an FPT algorithm working in $\mathcal{O}^*(2^k)$ time via the technique

of iterative compression [115].

We would like to remark here that algorithmic problems on tournaments are also interesting for they model various ranking problems naturally appearing in the theory of social choice. For example, one of the motivations of the study of Karpinski and Schudy [205] was obtaining a subexponential parameterized algorithm for the well-studied Kemeny Ranking Aggregation problem that reduces to the weighted variant of FAST. Since these applications of algorithms on tournaments are not in the scope of this thesis, we refer, e.g., to the work of Karpinski and Schudy [205] or of Kenyon-Mathieu and Schudy [214] for more discussion on this topic.

5.1.4 Our results and techniques

The material contained in this part originates in the results obtained in the following three papers:

- *Jungles, bundles, and fixed-parameter tractability*, co-authored with Fedor V. Fomin, and presented at the 24th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013 [146];
- *Computing cutwidth and pathwidth of semi-complete digraphs via degree orderings*, presented at the 30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, Kiel, Germany, February 27 - March 2, 2013 [269];
- *Subexponential parameterized algorithm for computing the cutwidth of a semi-complete digraph*, co-authored with Fedor V. Fomin, and to be presented at the 19th Annual European Symposium, ESA 2013, Sophia Antipolis, France, September 2-4, 2013 [147];

In this series, we addressed the plethora of open questions about fixed-parameter tractability of topological problems in semi-complete digraphs that arose from the containment theory of CFKSS. In particular, we investigated possibility of designing efficient, in terms of dependence on the parameter, fixed-parameter tractable algorithms for testing containment relations and computing the width measures. Such algorithms are most useful later as subroutines for algorithms for other problems, especially given the well-quasi-ordering results of Chudnovsky and Seymour [74], and of Kim and Seymour [216]. Figure on page 68 contains a summary of the findings. This work is meant to be a comprehensive compilation of the obtained results. We now present a chronological record of the findings in order to give the reader an insight into our line of reasoning.

The results of [146]. The first obstacle that needed to be overcome was the lack of an FPT approximation algorithm for pathwidth. Note that this was also the main reason why the algorithm for testing topological containment was not fixed-parameter tractable. This gap has been bridged in [146]. There, we have presented an FPT approximation algorithm for pathwidth that, given a semi-complete digraph T on n vertices and an integer k, either finds a path decomposition of T of width $\mathcal{O}(k^2)$ or provides an obstacle for admitting a path decomposition of width at most k, and runs in $2^{\mathcal{O}(k \log k)} \cdot n^3 \log n$ time. The approach was based on replacing the crucial part of the algorithm of Fradkin and Seymour [153] that was implemented by a brute-force enumeration, by a more refined argument based on the colour coding technique of Alon et al. [13]. Since we have later found even faster algorithms, in this work we omit the description of the approximation algorithm for pathwidth presented in [146].

Having an FPT approximation algorithm for pathwidth, we were able to show an FPT algorithm testing topological containment. Note here that to obtain this result one needs to implement the

Problem	Previous results (CFKSS)	This work
Cutwidth approximation	$\mathcal{O}(n^3)$ time, width $\mathcal{O}(k^2)$ [72]	$\mathcal{O}(n^2)$ time, width $\mathcal{O}(k^2)$ (Thm 68)
Cutwidth exact	$f(k) \cdot n^3$ time, non-uniform, non-constructive [72, 74]	$2^{\mathcal{O}(\sqrt{k \log k})} \cdot n^2$ time (Thm 88)
Pathwidth approximation	$\mathcal{O}(n^{\mathcal{O}(k)})$ time, width $\mathcal{O}(k^2)$ [153]	$\mathcal{O}(kn^2)$ time, width $6k$ (Thm 65)
Pathwidth exact	$\mathcal{O}(n^{m(k)})$ time, non-uniform, non-constructive [216]	$2^{\mathcal{O}(k \log k)} \cdot n^2$ time (Thm 66)
Topological containment	$\mathcal{O}(n^{m(\lvert H\rvert)})$ time [153]	$2^{\mathcal{O}(\lvert H\rvert \log \lvert H\rvert)} \cdot n^2$ time (Thm 94)
Immersion	$f(\lvert H\rvert) \cdot n^3$ time [72]	$2^{\mathcal{O}(\lvert H\rvert^2 \log \lvert H\rvert)} \cdot n^2$ time (Thm 95)
Minor	$\mathcal{O}(n^{m(\lvert H\rvert)})$ time	$2^{\mathcal{O}(\lvert H\rvert \log \lvert H\rvert)} \cdot n^2$ time (Thm 96)
Rooted Immersion	-	$f(\lvert H\rvert) \cdot n^3$ time (Thm 106)
Edge Disjoint Paths	$f(k) \cdot n^5$ time [152]	$f(k) \cdot n^3$ time (Thm 106)
Vertex Disjoint Paths	$\mathcal{O}(n^{m(k)})$ time [73]	-

Figure 5.1: Comparison of previously known algorithms and the results of this thesis. The algorithms for cutwidth and pathwidth take as input a semi-complete digraph on n vertices and an integer k. The approximation algorithms can output a decomposition of larger width (guarantees are in the corresponding cells) or conclude that a decomposition of width at most k does not exist. The exact algorithms either construct a decomposition of width at most k or conclude that this is impossible. We remark that the XP algorithm for testing the minor relation has not been stated explicitely in any of the previous works, but it follows from them easily.

dynamic programming routine working on a path decomposition in FPT time, while the original routine of Fradkin and Seymour was not fixed-parameter tractable. This, however, turned out to be less challenging, and it also follows from known tools on model checking $\mathbf{MSO}_1$ on digraphs of bounded cliquewidth. We give an introduction to tools borrowed from logic in Section 5.3, and design an explicit dynamic programming routine in Section 7.4 in order to give precise guarantees on the time complexity of the topological containment test. Algorithms for immersion and minor testing can be designed in a similar manner. Fixed-parameter tractability of immersion and minor testing opens possibilities for proving meta-theorems of more general nature via the well-quasi-ordering results of Chudnovsky and Seymour [74], and of Kim and Seymour [216]. We give an example of such a meta-theorem in Section 7.2.

Another problem that we were able to address using the FPT approximation for pathwidth, was the Rooted Immersion problem, whose parameterized complexity was left open by Fradkin and Seymour [152]. Unlike Fradkin and Seymour, we have approached the problem via pathwidth instead of cutwidth, having observed beforehand that a dynamic programming routine testing immersion on a path decomposition of small width can be designed similarly to topological containment. Hence, after running the approximation algorithm for pathwidth, either we can apply the dynamic programming on the obtained decomposition, or we are left with an obstacle for pathwidth, called a *triple*, which is more powerful than the obstacles for cutwidth used by Fradkin and Seymour. Similarly to Fradkin and Seymour, we design an irrelevant vertex rule on a triple, that is, in polynomial time we identify a vertex that can be safely removed from the triple without changing answer to the problem. Then we restart the algorithm. Observe that the algorithm can make at most n iterations before solving the problem by applying dynamic programming, since there are only n vertices in the digraph. All in all, this gives an algorithm for Rooted Immersion working in $f(|H|) \cdot n^4 \log n$ time for some elementary function f. We describe this algorithm in Section 7.3.

The results of [269]. In the next paper [269] we have discovered a completely different approach, dubbed the *degree ordering approach*, that enabled us to design much more efficient algorithms both for pathwidth and for cutwidth. Using the new technique we were able to reprove the structural theorems for both parameters in a unified and simplified way, obtaining in both cases polynomial-time approximation algorithms and, in case of pathwidth, even a constant-factor approximation. The technique could be also used to develop FPT exact algorithm for both width measures with single-exponential dependence of the running time on the optimal width. Moreover, we were able to trim the dependence of the running time on the size of the tested digraph to single-exponential in all of the containment tests. All the algorithms obtained via the approach of degree orderings have quadratic dependence on the number of vertices of the given semi-complete digraph T; note that this is *linear* in the input size. Since the degree ordering approach will be the main tool used in the following chapters, let us spend some space on explaining its main points.

The crucial observation of the degree ordering approach can be intuitively formulated as follows: any ordering of vertices with respect to increasing outdegrees is a good approximation of the order in which the vertices appear in some path decomposition close to optimal. In fact, for cutwidth this is true even in formal sense. We proved that any outdegree ordering of vertices of a semi-complete digraph T has width at most $\mathcal{O}(\mathbf{ctw}(T)^2)$, hence we have a trivial approximation algorithm that sorts the vertices with respect to outdegrees. As far as computing cutwidth exactly is concerned, the developed set of tools gives raise to an algorithm testing whether the cutwidth of a given semi-complete digraph on n vertices is at most k, working in $2^{\mathcal{O}(k)} \cdot n^2$ time. Shortly

speaking, we scan through the outdegree ordering with a dynamic program, maintaining a bit mask of length $\mathcal{O}(k)$ denoting which vertices of an appropriate interval of the ordering are contained in a constructed prefix of an optimal ordering. Since we later have found an even faster exact algorithm for cutwidth of a semi-complete digraph, in this work we omit the description of this algorithm.

The case of pathwidth, which was of our main interest, is more complicated. The intuitive outdegree ordering argument can be formalized as follows: we prove that existence of $4k + 2$ vertices with outdegrees pairwise not differing by more than k already forms an obstacle for admitting a path decomposition of width at most k; we remark here that in [269] we used a weaker bound $5k+2$, which led to worse approximation ratio for the approximation algorithm. We call this obstacle a *degree tangle*. Hence, any outdegree ordering of vertices of a given semi-complete digraph T of small pathwidth must be already quite spread: it does not contain larger clusters of vertices with similar outdegrees. This spread argument is crucial in all our reasonings, and shows that finding new structural obstacles can significantly improve our understanding of the problem.

Similarly as in the case of the exact algorithm for cutwidth, both the approximation and the exact algorithm for pathwidth use the concept of scanning through the outdegree ordering with a *window* — an interval in the ordering containing $4k$ vertices. By the outdegree spread argument, at each point we know that the vertices on the left side of the window have outdegrees smaller by more than k that the ones on the right side; otherwise we would have a too large degree tangle. For approximation, we construct the consecutive bags by greedily taking the window and augmenting this choice with a small coverage of arcs jumping over it. The big gap between outdegrees on the left and on the right side of the window ensures that nonexistence of a small coverage is also an evidence for not admitting a path decomposition of small width. The obtained approximation ratio is 6; in the original paper we claimed approximation ratio 7 because of a weaker bound on the size of a degree tangle that is an obstacle for pathwidth k. For the exact algorithm, we identify a set of $\mathcal{O}(k^2)$ vertices around the window, about which we can safely assume that the bag is contained in it. Then we run a similar dynamic programming algorithm as in the case of cutwidth.

The most technical part in the approximation and exact algorithms for pathwidth is the choice of vertices outside the window to cover the arcs jumping over it. It turns out that this problem can be expressed as trying to find a small vertex cover in an auxiliary bipartite graph. However, in order to obtain a feasible path decomposition we cannot choose the vertex cover arbitrarily — it must behave consistently as the window slides through the ordering. To this end, in the approximation algorithm we use a 2-approximation of the vertex cover based on the theory of matchings in bipartite graphs. In the exact algorithm we need more restrictions, as we seek a subset that contains *every* sensible choice of the bag. Therefore, we use an $\mathcal{O}(OPT)$-approximation of vertex cover based on the classical kernelization routine for the problem of Buss [54], which enables us to use stronger arguments to reason which vertices can be excluded from consideration.

We also observe that the obtained set of obstacles for pathwidth have much better properties than the ones introduced by Fradkin and Seymour [153]. We show that there is a constant multiplicative factor relation between the sizes of obstacles found by the algorithm and the minimum size of a digraph that cannot be embedded into them. Hence, in all the containment tests we just need to run the pathwidth approximation with parameter $\mathcal{O}(|H|)$, and in case of finding an obstacle just provide a positive answer. This trims the dependence of running time of all the containment tests to single exponential in terms of the size of the tested subgraph, compared to multiple-exponential following from the previous work. In addition, if in the algorithm for Rooted Immersion one substitutes the approximation algorithm for pathwidth of [146] by the newer algorithm of [269],

the polynomial factor gets reduced from $n^4 \log n$ to n^3.

The results of [147]. Finally, in the most recent paper of the series [147] we have observed that the cutwidth of a semi-complete digraph can be computed exactly even faster — in subexponential parameterized time. To achieve this running time, we have adopted the technique of *k-cuts*, developed earlier together with Fedor Fomin, Stefan Kratsch, Marcin Pilipczuk and Yngve Villanger [137] in the context of clustering problems; the results of [137] are presented in Chapter 8 in Part III. Shortly speaking, the single-exponential dynamic programming algorithm presented in [269] searched through a space of states of size $2^{\mathcal{O}(k)} \cdot n$. The new observation is that this search space can be in fact restricted tremendously to subexponential size by a more refined combinatorial analysis of the problem. The main idea is to relate sensible states of the dynamic program (i.e., the aforementioned k-cuts) to *partition numbers*: the partition number $p(k)$ is the number of different multisets of positive integers summing up to k. The subexponential asymptotics of partition numbers have been very well understood from the point of view of enumerative combinatorics [122, 186], and we can use the results obtained there directly in our setting in order to bound the number of states of the dynamic program. All in all, this number turns out to be bounded by $2^{\mathcal{O}(\sqrt{k \log k})} \cdot (n+1)$, and the whole algorithm can be implemented in $2^{\mathcal{O}(\sqrt{k \log k})} \cdot n^2$ time.

As a byproduct of the approach taken, we also obtain a new algorithm for Feedback Arc Set in semi-complete digraphs with running time $\mathcal{O}(2^{c\sqrt{k}} \cdot k^{\mathcal{O}(1)} \cdot n^2)$ for $c = \frac{2\pi}{\sqrt{3} \cdot \ln 2} \leq 5.24$. The new algorithm is simpler than the aforementioned algorithms of Feige [126] and of Karpinski and Schudy[205], both also working in $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k})})$ time. It is also worth mentioning that the explicit constant in the exponent obtained using our approach is much smaller than the constants in the algorithms of Feige and of Karpinski and Schudy; however, optimizing these constants was not the purpose of these works.

Lastly, it appears that our approach can be also applied to other layout problems in semi-complete digraphs. For example, we consider a natural adaptation of the Optimal Linear Arrangement problem [71, 111] to the semi-complete setting, and we prove that one can compute in $2^{\mathcal{O}(k^{1/3} \cdot \sqrt{\log k})} \cdot n^2$ time an ordering of cost at most k, or conclude that it is impossible (in Section 6.3.2 we give a precise definition of Optimal Linear Arrangement in semi-complete digraphs). Although such a low complexity may be explained by the fact that the optimal cost may be even cubic in the number of vertices, we find it interesting that results of this kind can be also obtained by making use of the developed techniques.

We remark that in the version presented in [147], the running times of the algorithms actually contained larger polynomial factors hidden in the $\mathcal{O}^*(\cdot)$ notation. In this work we apply the ideas from [269] to obtain algorithms that are at the same time subexponential, and quadratic in terms of the number of vertices of the input semi-complete digraph.

Outline. Section 5.2 is devoted to extended preliminaries. After explaining notation and basic facts about semi-complete digraphs, we give formal definitions of containment notions, of width measures, and prove a number of results relating them. In particular, we show how the width measures relate to each other, and which width measures are closed under which containment relations. Apart from cutwidth and pathwidth that are of main interest in this work, we introduce also cliquewidth. The reason is that relations between cutwidth, pathwidth and cliquewidth in semi-complete digraphs are crucial for Section 5.3, where we gather the tools borrowed from logic that will be used later on.

In Chapter 6 we present the algorithms for computing pathwidth and cutwidth of a semi-complete digraph. We begin with explaining the set of obstacles for these width measures in Section 6.1, which gives foundations for the degree ordering approach. Then the approximation and exact algorithms for pathwidth are explained in Section 6.2, and the approximation and exact algorithms for cutwidth in Section 6.3. The algorithms for Feedback Arc Set in semi-complete digraphs and for Optimal Linear Arrangement are presented along with the exact algorithm for cutwidth, since they use the same set of tools. In Section 6.4 we give some concluding remarks on problems concerning computational complexity of cutwidth and pathwidth, and present a number of open problems that arise from this work.

In Chapter 7 we utilize the results of Chapter 6 to design algorithms for topological problems in semi-complete digraphs. We begin by presenting the algorithms testing containment relations in Section 7.1. Then, in Section 7.2 we give an example of a meta-theorem that can be proven using a combination of our algorithms and the well-quasi-ordering results of Chudnovsky and Seymour [74], and of Kim and Seymour [216]. In Section 7.3 we give an algorithm for the Rooted Immersion problem. Section 7.4 contains descriptions of explicit dynamic programming routines for topological containment and immersion working on a path decomposition, which are used in Sections 7.1 and 7.3. Even though general construction of such routines follows directly from the tools borrowed from logic, we need to construct them explicitely in order to give precise upper bounds on the running times of the obtained algorithms. Finally, in Section 7.5 we conclude and give some open problems.

5.2 Preliminaries

5.2.1 Folklore and simple facts

In this section we provide some simple facts about semi-complete digraphs and tournaments that will be used in this chapter. We begin with some observations on the out- and indegrees in semi-complete digraphs. Let T be a semi-complete digraph. Note that for every $v \in V(T)$ we have that $d^+(v) + d^-(v) \geq |V(T)| - 1$, and that the equality holds for all $v \in V(T)$ if and only if T is a tournament.

Lemma 19. *Let T be a semi-complete digraph. Then the number of vertices of T with outdegrees at most d is at most $2d + 1$.*

Proof. Let A be the set of vertices of T with outdegrees at most d, and for the sake of contradiction assume that $|A| > 2d + 1$. Consider semi-complete digraph $T[A]$. By a simple degree-counting argument, in every semi-complete digraph S there is a vertex of outdegree at least $\frac{|V(S)|-1}{2}$. Hence, $T[A]$ contains a vertex with outdegree larger than d. As outdegrees in $T[A]$ are not smaller than in T, this is a contradiction with the definition of A. □

Lemma 20. *Let T be a semi-complete digraph and let x, y be vertices of T such that $d^+(x) > d^+(y) + \ell$. Then there exist at least ℓ vertices that are both outneighbors of x and inneighbors of y and, consequently, ℓ vertex-disjoint paths of length 2 from x to y.*

Proof. Let $\alpha = d^+(y)$. We have that $d^-(y) + d^+(x) \geq |V(T)| - 1 - \alpha + \alpha + \ell + 1 = |V(T)| + \ell$. Hence, by the pigeonhole principle there exist at least ℓ vertices of T that are both outneighbors of x and inneighbors of y. □

We now proceed to a slight generalization of a folklore fact that every strongly connected tournament has a hamiltonian cycle; see for example Problem 3 from the finals (third round) of XLI Polish Mathematical Olympiad [14]. In fact, this observation holds also in the semi-complete setting, and we include its proof for the sake of completeness.

Lemma 21. *A semi-complete digraph T has a hamiltonian cycle if and only if it is strongly connected.*

Proof. Necessary condition being trivial, we proceed to the proof that every strongly connected semi-complete digraph T has a hamiltonian cycle. We proceed by induction on $|V(T)|$. The base cases when T has one or two vertices are trivial, so we proceed with the assumption that $|V(T)| > 2$.

Let v be any vertex of T and let $T' = T \setminus v$. Let $T_1, T_2, \ldots, T_p$ be the strongly connected components of T'. Note that since T' is semi-complete, the directed acyclic graph of its strongly connected components must be also semi-complete, hence it must be a transitive tournament. Without loss of generality let $T_1, T_2, \ldots, T_p$ be ordered as in the unique topological ordering of this transitive tournament, i.e., for every $v_1 \in V(T_i)$ and $v_2 \in V(T_j)$ where $i \neq j$, we have that $(v_1, v_2) \in E(T')$ if and only if $i < j$. Since $T_1, T_2, \ldots, T_p$ are strongly connected, by inductive hypothesis let $C_1, C_2, \ldots, C_p$ be hamiltonian cycles in $T_1, T_2, \ldots, T_p$, respectively.

Observe that there must be some vertex $v' \in V(T_1)$ such that $(v, v') \in E(T)$, as otherwise $(V(T_1), \{v\} \cup \bigcup_{i=2}^{p} V(T_i))$ would be a partition of $V(T)$ such that all the arcs between the left side and the right side of the partition are directed from the left to the right; this would be a contradiction with T being strongly connected. A symmetric reasoning shows that there exists some vertex $v'' \in V(T_p)$ such that $(v'', v) \in E(T)$.

We now distinguish two cases. In the first case we assume that $p = 1$. Let $v_1, v_2, \ldots, v_{n-1}$ be the vertices of $V(T') = V(T_1)$ ordered as on cycle C_1, i.e., $(v_i, v_{i+1}) \in E(T')$ where index i behaves cyclically. Without loss of generality assume that $v_1 = v''$. We claim that there are two vertices v_i, v_{i+1} (where again i behaves cyclically) such that $(v, v_{i+1}) \in E(T)$ and $(v_i, v) \in E(T)$. If every vertex v_i is a tail of an arc directed towards v, then this claim is trivial: we just take index i such that $v_{i+1} = v'$. Otherwise there are some vertices that are not tails of arcs directed towards v, and let $i+1$ be the smallest index of such a vertex. Note that by the assumption that $v_1 = v''$ we have that $i + 1 > 1$. Since T is semi-complete, it follows that $(v, v_{i+1}) \in E(T)$. By the minimality of $i + 1$ and the fact that $i + 1 > 1$, it follows that $(v_i, v) \in E(T)$, which proves that vertices v_i, v_{i+1} have the claimed property. We now can construct a hamiltonian cycle C for the whole digraph T by inserting v between v_i and v_{i+1} in C_1; note that here we use the fact that $|V(T)| > 2$ so that v_i and v_{i+1} are actually two different vertices.

Now assume that $p > 1$. We construct a hamiltonian cycle C for the whole T by concatenating cycles $C_1, C_2, \ldots, C_p$. To construct C, take first v and then place v' followed by the whole cycle C_1 traversed from v' to the predecessor of v'. Then proceed to an arbitrarily chosen vertex of C_2 and traverse the whole cycle C_2 from this vertex up to its predecessor. Continue in this manner through the consecutive components, but when considering C_p, instead of choosing an arbitrary vertex to begin with, choose the successor of v'' on C_p so that after traversing C_p we arrive at v'' that is a tail of an arc directed to v. It is easy to observe that C constructed in this manner is indeed a hamiltonian cycle: it follows from the fact that $(v, v'), (v'', v) \in E(T)$, and for every two consecutive strongly connected components T_i, T_{i+1}, there is an arc from every vertex of the first component to every vertex of the second component. □

5.2.2 Definitions of containment relations

In this section we introduce formally the containment notions that will be of our interest. We start with the *immersion* and *topological containment* relations, which are direct analogues of the classical undirected versions. Then we proceed to the notion of *minor*, for which one needs to carefully describe how the undirected notion is translated to the directed setting.

Let H, G be digraphs. We say that mapping η is a *model* of H in G, if the following conditions are satisfied:

- for every vertex $v \in V(H)$, $\eta(v)$ is a subset of $V(G)$;
- for every arc $(u, v) \in E(H)$, $\eta((u, v))$ is a directed path leading from some vertex of $\eta(u)$ to some vertex of $\eta(v)$.

By imposing further conditions on the model we obtain various containment notions for digraphs. If we require that η:

- maps vertices of H to pairwise different singletons of vertices of G,
- and the paths in $\eta(E(H))$ are internally vertex-disjoint,

then we obtain the notion of *topological containment*. In this case we say that η is an *expansion* of H in G, and we say that H is a *topological subgraph* of G. As the images of vertices are singleton sets, we may think that η simply maps vertices of H to vertices of G. If we relax the condition on paths in $\eta(E(H))$ from being internally vertex-disjoint to being arc-disjoint, we arrive at the notion of *immersion*; then η is an *immersion* of H into G. Clearly, every expansion is also an immersion, so if G topologically contains H, then it contains H as an immersion as well.

Sometimes in the literature this notion is called *weak immersion* to distinguish it from *strong immersion*, where each image of an arc is additionally required not to pass through images of vertices not being the endpoints of this arc. In this work we are interested only in (weak) immersions, as we find the notion of strong immersion not well-motivated enough to investigate all the technical details that arise when considering both definitions at the same time and discussing slight differences between them.

Finally, we proceed to the notion of a *minor*. For this, we require that η

- maps vertices of H to pairwise disjoint sets of vertices of G that moreover induce strongly connected subdigraphs,
- and maps arcs from $E(H)$ to arcs of $E(G)$ in such a manner that $\eta((u, v))$ is an arc from a vertex of $\eta(u)$ to a vertex of $\eta(v)$, for every $(u, v) \in E(H)$.

We then say that η is a *minor model* of H in G. In other words, we naturally translate the classical notion from the undirected graphs to the directed graphs by replacing the connectivity requirement with strong connectivity.

The notion of a minor of digraphs was introduced by Kim and Seymour [216], and we would like to remark that the original definition is more general as it handles also the case of digraphs with multiple arcs and loops. As in this work we are interested in simple digraphs only, we will work with this simplified definition, and we refer a curious reader to the work of Kim and Seymour [216] for more details on the general setting.

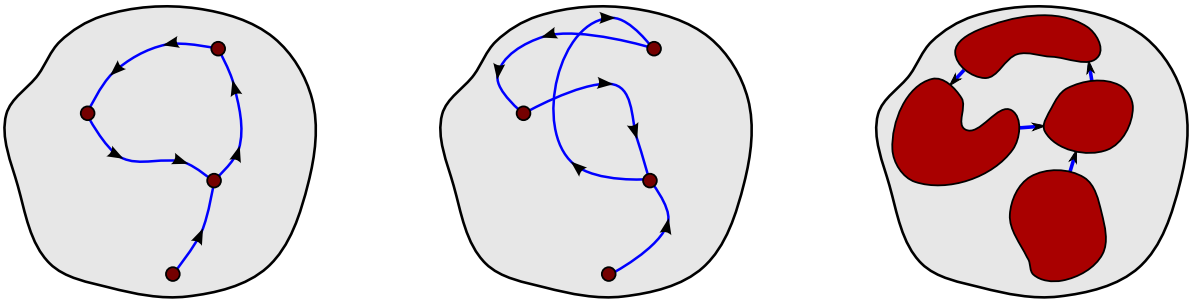

Figure 5.2: Expansion, immersion, and a minor model of the same 4-vertex digraph H in a larger digraph G.

The introduced minor order is not directly stronger or weaker than the immersion or topological containment orders. In particular, contrary to the undirected setting, it is not true that if G contains H as a topological subgraph, then it contains H as a minor; as a counterexample take G being a triangle (a tournament being a directed cycle of length 3) and H being a complete digraph on 2 vertices (two vertices plus two arcs in both directions).

However, in the semi-complete setting the minor containment testing may be conveniently Turing-reduced to topological containment testing using the following lemma. In the following, we use the notion of *constrained topological containment*. We say that a digraph H is topologically contained in G *with constraints* $F \subseteq E(H)$, if all the images of arcs of F are of length 1, i.e., they are just single arcs in G.

Lemma 22. *There is an algorithm that, given a digraph H, in $2^{\mathcal{O}(|H|\log|H|)}$ time computes a family $\mathcal{F}_H$ of pairs (D, F) where D is a digraph and $F \subseteq E(D)$, with the following properties:*

(i) $|\mathcal{F}_H| \leq 2^{\mathcal{O}(|H|\log|H|)}$;

(ii) $|D| \leq 5|H|$ *for each* $(D, F) \in \mathcal{F}_H$;

(iii) for any semi-complete digraph T, H is a minor of T if and only if there is at least one pair $(D, F) \in \mathcal{F}_H$ such that D is topologically contained in T with constraints F.

Proof. We present first how the family $\mathcal{F}_H$ is constructed. For every vertex $u \in V(H)$, choose a number $p(u)$ between 1 and $d(u)$. Construct a directed cycle C_u of length $p(u)$ (in case $p(u) = 1$ take a single vertex without a loop), and let $u_1, u_2, \ldots, u_{p(u)}$ be vertices of C_u in this order. For every arc $(u, v) \in E(H)$, choose two integers $i, j = i((u, v)), j((u, v))$ such that $1 \leq i \leq p(u)$ and $1 \leq j \leq p(v)$, and add an arc (u_i, v_j). Family $\mathcal{F}_H$ consists of all the digraphs constructed in this manner, together with the sets of all the arcs not contained in cycles $C(u)$ as constraints.

Property (ii) follows directly from the construction, so let us argue that property (i) is satisfied. For every vertex u we have at most $d(u)$ choices of $p(u)$ and at most $d(u)^{d(u)}$ choices for integers $i((u, v))$ and $j((w, u))$ for $(u, v), (w, u) \in E(H)$. Hence, in total we have at most $\prod_{u \in V(H)} d(u)^{d(u)+1} = \prod_{u \in V(H)} 2^{\mathcal{O}(d(u)\log d(u))}$ choices. Since function $t \to t \log t$ is convex and $\sum_{u \in V(H)} d(u) = \mathcal{O}(|E(H)|)$, in total we will construct at most $2^{\mathcal{O}(|H|\log|H|)}$ digraphs. As each digraph from H is constructed in polynomial time, the running time of the algorithm also follows.

We are left with proving that property (iii) is satisfied as well. Assume first that we are given a semi-complete digraph T, and there is a pair $(D, F) \in \mathcal{F}_H$ such that T contains expansion η of some D, where the arcs of F are mapped to single arcs in T. Since cycles $C(u)$ in D are strongly

connected, so do their images in η. Hence, to construct a minor model of H in G, we can simply map every vertex of $u \in V(H)$ to $V(\eta(C(u)))$. Existence of appropriate arcs modeling arcs of $E(H)$ follows from the fact that arcs of F are mapped to single arcs in T in η.

Assume now that G admits a minor model η of H. For every set $\eta(u)$ for $u \in V(H)$, construct a hamiltonian cycle $C_0(u)$ in $T[\eta(u)]$ using Lemma 21. Then define a digraph D by taking T, and

- removing all the vertices not participating in any cycle $C_0(u)$,
- removing all the arcs not participating in any cycle $C_0(u)$ and not being images of arcs of H in η,
- and contracting all the paths of vertices with degrees 2 on cycles $C_0(u)$ to single arcs.

Observe that since at most $d(u)$ vertices of $C_0(u)$ are incident to images of arcs of H incident to u, then cycle $C_0(u)$ after contractions have length at most $d(u)$. Therefore, it can be easily seen that the obtained digraph D is enumerated when constructing family $\mathcal{F}_H$, and moreover it is enumerated together with the set of images of arcs of H in η as constraints. Construction of D ensures that D is topologically contained in G with exactly these constraints. □

One of the motivations of work of Chudnovsky, Fradkin, Kim, Scott,and Seymour, was finding containment notions that are well-quasi orders on the class of semi-complete digraphs. It appears that both immersion and minor orders have this property. As far as topological containment is concerned, Kim and Seymour [216] observe that it does not form a well-quasi-ordering.

Theorem 23 ([74]). *The immersion ordering is a well-quasi-ordering of the class of semi-complete digraphs.*[1]

Theorem 24 ([216]). *The minor ordering is a well-quasi-ordering of the class of semi-complete digraphs.*

Finally, we say that $\mathbf{G} = (G; v_1, \ldots, v_h)$ is a *rooted* digraph if G is digraph and $v_1, \ldots, v_h$ are pairwise different vertices of $V(G)$. The notions of immersion and topological containment can be naturally generalized to rooted digraphs. Immersion η is an immersion from a rooted digraph $\mathbf{H} = (H; u_1, \ldots, u_h)$ to a rooted digraph $\mathbf{G} = (G; v_1, \ldots, v_h)$ if additionally $\eta(u_i) = v_i$ for $i \in \{1, \ldots, h\}$, that is, the immersion preserves the roots. Such an immersion is called an $\mathbf{H}$-*immersion* or a *rooted immersion*. In the same manner we may define $\mathbf{H}$-*expansions* or *rooted expansions*.

5.2.3 Width parameters

In this section we introduce formally the width notions of digraphs that will be used in this chapter: cutwidth, pathwidth, and cliquewidth. As far as cutwidth and pathwidth are of our prime interest, we introduce cliquewidth for the sake of introducing meta-tools concerning model checking Monadic Second-Order logic that will make some of our later arguments cleaner and more concise. We first explain each of the parameters separately, and then proceed to proving inequalities between them.

[1] Chudnovsky and Seymour state the result for tournaments only, but the proof works actually also in the semi-complete setting; cf. [216].

Cutwidth

The notion of cutwidth of digraphs resembles the classical definition in the undirected setting, with an exception that only arcs directed forwards in the ordering contribute to the cut function.

Definition 25. *Given a digraph $G = (V, E)$ and an ordering π of V, let $\pi[\alpha]$ be the first α vertices in the ordering π. The* width *of π is equal to $\max_{0 \leq \alpha \leq |V|} |E(\pi[\alpha], V \setminus \pi[\alpha])|$; the* cutwidth *of G, denoted $\mathbf{ctw}(G)$, is the minimum width among all orderings of V.*

Note that any transitive tournament T has cutwidth 0: we simply take the reversed topological ordering of T. It appears that cutwidth is closed under taking immersions, i.e., if H is an immersion of G then $\mathbf{ctw}(H) \leq \mathbf{ctw}(G)$.

Lemma 26. *Let H, G be digraphs and assume that H can be immersed into G. Then $\mathbf{ctw}(H) \leq \mathbf{ctw}(G)$.*

Proof. Let σ be an ordering of $V(G)$ of width $\mathbf{ctw}(G)$ and let η be immersion of H into G. Define ordering σ' of $V(H)$ by setting $u <_{\sigma'} v$ if and only if $\eta(u) <_{\sigma} \eta(v)$. We claim that σ' has width at most $\mathbf{ctw}(G)$. Indeed, take any prefix $\sigma'[t']$ for $0 \leq t' \leq |V(H)|$ and corresponding suffix $V(H) \setminus \sigma'[t']$. By the definition of σ' we can chose a number t, $0 \leq t \leq |V(G)|$ such that $\eta(\sigma'[t']) \subseteq \sigma[t]$ and $\eta(V(H) \setminus \sigma'[t']) \subseteq V(G) \setminus \sigma[t]$. Now consider any arc $(u, v) \in E(H)$ such that $u \in \sigma'[t']$ and $v \in V(H) \setminus \sigma'[t']$; we would like to prove that the number of such arcs is at most $\mathbf{ctw}(G)$. However, $\eta((u, v))$ is a directed path from $\eta(u) \in \sigma[t]$ to $\eta(v) \in V(G) \setminus \sigma[t]$. All these paths are edge-disjoint and contain at least one arc in $E(\sigma[t], V(G) \setminus \sigma[t])$. As the number of such arcs is at most $\mathbf{ctw}(G)$, the lemma follows. □

Pathwidth

Similarly to cutwidth, also the notion of pathwidth is a direct translation of the definition in the undirected setting, again bearing in mind the intuition that only arcs directed forwards in the decomposition are contributing to the width.

Definition 27. *Given a digraph $G = (V, E)$, a sequence $W = (W_1, \ldots, W_r)$ of subsets of V is a* path decomposition of G *if the following conditions are satisfied:*

(i) $\bigcup_{1 \leq i \leq r} W_i = V$;

(ii) $W_i \cap W_k \subseteq W_j$ *for* $1 \leq i < j < k \leq r$;

(iii) $\forall\ (u, v) \in E$, *either* $u, v \in W_i$ *for some* i *or* $u \in W_i$, $v \in W_j$ *for some* $i > j$.

We call $W_1, \ldots, W_r$ the bags *of the path decomposition. The* width *of a path decomposition is equal to $\max_{1 \leq i \leq r}(|W_i| - 1)$; the* pathwidth *of G, denoted $\mathbf{pw}(G)$, is the minimum width among all path decompositions of G.*

We would like to remark that conditions (i) and (ii) are equivalent to saying that for every vertex v, the set of bags containing v form a nonempty interval in the path decomposition. By I_v^W we denote this interval in W, treated as an interval of indices; in the notation we drop the decomposition W whenever it is clear from the context. Then condition (iii) is equivalent to saying that if (u, v) is an arc, then it cannot occur that I_u ends in the decomposition W before I_v starts. We use this equivalent definition interchangeably with the original one.

Note that Definition 27 formally allows having duplicate bags in the decomposition, thus making it possible for a path decomposition to have unbounded length. However, of course we may remove any number of consecutive duplicate bags while constructing the decomposition. Hence, we will implicitly assume that in all the path decompositions any two consecutive bags are different. Then, for any two consecutive bags W_i, W_{i+1} there exists a vertex $v \in V(T)$ such that interval I_v either ends in i or begins in $i+1$. Since there are $2|V(T)|$ beginnings and ends of intervals I_v for $v \in V(T)$ in total, we infer that any path decomposition W of T which does not contain any two consecutive equal bags has length at most $2|V(T)|+1$.

Note that any transitive tournament T has pathwidth 0: we can construct a decomposition W of width 0 by taking singletons of all the vertices and ordering them according to the reversed topological ordering of T. Again, it appears that pathwidth is closed both under taking minors and topological subgraphs, i.e., if H is a topological subgraph or a minor of G then $\mathbf{pw}(H) \leq \mathbf{pw}(G)$. The second observation was also noted by Kim and Seymour [216]; we include both of the proofs for the sake of completeness.

Lemma 28. *Let H, G be digraphs and assume that H is a topological subgraph of G. Then $\mathbf{pw}(H) \leq \mathbf{pw}(G)$.*

Proof. Let W be a path decomposition of G of width $\mathbf{pw}(G)$, and let η be an expansion of H in G. Take any $(u,v) \in E(H)$ and examine path $Q = \eta(u,v)$. For two vertices $x, y \in V(Q)$, $x \neq y$, we say that x and y *communicate* if $I_x \cap I_y \neq \emptyset$. By a *communication graph* $C(Q)$ we mean an undirected graph defined on $V(Q)$, where two vertices are connected via an edge if and only if they communicate.

We say that an arc $(u,v) \in E(H)$ is *problematic*, if interval $I_{\eta(v)}$ starts in W after $I_{\eta(u)}$ ends. We claim now that if (u,v) is a problematic arc then $\eta(u)$ and $\eta(v)$ are in the same connected component of $C(Q)$ for $Q = \eta((u,v))$. Assume otherwise. Let $C_1, C_2, \ldots, C_p$ be the connected components of $C(Q)$, where without loss of generality $\eta(u) \in V(C_1)$ and $\eta(v) \in V(C_2)$. By the definition of communication, for every $i = 1, 2, \ldots, p$ we have that $I_{C_i} = \bigcup_{x \in V(C_i)} I_x$ is an interval, and these intervals are pairwise disjoint. Since $I_{\eta(v)}$ starts after $I_{\eta(u)}$ finishes, we have that I_{C_2} is situated after I_{C_1} in the decomposition. Now consider consecutive arcs (x, x') of the path Q, and observe that by the definition of a path decomposition, it follows that either x and x' belong to the same component C_i, or the interval of the component to which x' belongs must be situated in the decomposition before the interval of the component of x. Hence, all the intervals $I_{C_2}, I_{C_3}, \ldots, I_{C_p}$ must be in fact situated before I_{C_1} in the path decomposition, which is a contradiction with the fact that I_{C_2} is situated after.

Let $L(Q)$ be the set of internal vertices of any path connecting $\eta(u)$ and $\eta(v)$ in $C(Q)$. Note that by the definition of communication, $I_{\eta(u)} \cup \bigcup_{x \in L(Q)} I_x$ is an interval in the path decomposition, and sets $L(\eta(a))$ are disjoint for different problematic arcs $a \in E(H)$.

We now construct a decomposition of the graph H. Take the decomposition W and:

- for every $u \in V(H)$, replace each occurrence of $\eta(u)$ by u in every bag of W;
- for every problematic arc $(u,v) \in E(H)$, replace each occurrence of any vertex from $L(\eta(u,v))$ with u (and remove duplicates, if necessary);
- remove all the remaining vertices of G from all the bags of decomposition W.

Let us denote the obtained sequence of bags by W'. Clearly, bags of W' contain only vertices of H, and they are of size at most $\mathbf{pw}(G)+1$, so it remains to prove that W' is a correct path decomposition of H.

Firstly, take any vertex $u \in V(H)$ and observe that the set of all the bags of W' containing u consists of bags with indices from $I_{\eta(u)}$ plus $\bigcup_{x \in L((u,v))} I_x$ for each problematic arc $(u,v) \in E(H)$. Since $I_{\eta(u)} \cup \bigcup_{x \in L((u,v))} I_x$ is an interval containing nonempty I_u for any such arc (u,v), we infer that the union of all these intervals is also a nonempty interval. This proves properties (i) and (ii).

Secondly, examine any arc $(u,v) \in E(H)$. If (u,v) is non-problematic, then either $I^W_{\eta(u)}$ and $I^W_{\eta(v)}$ overlap or $I^W_{\eta(u)}$ is situated after $I^W_{\eta(v)}$. Since $I^W_{\eta(u)} \subseteq I^{W'}_u$ and $I^W_{\eta(v)} \subseteq I^{W'}_v$, condition (iii) holds for non-problematic arcs. Assume then that (u,v) is a problematic arc. But then $I^W_{\eta(u)} \cup \bigcup_{x \in L((u,v))} I^W_x \subseteq I^{W'}_u$, and by the definition of $L((u,v))$ we have that $I^W_{\eta(u)} \cup \bigcup_{x \in L((u,v))} I^W_x$ has a nonempty intersection with $I^W_{\eta(v)}$. Since $I^W_{\eta(v)} \subseteq I^{W'}_v$, we have that $I^{W'}_u$ and $I^{W'}_v$ intersect. This proves property (iii). □

Lemma 29 (1.2 of [216]). *Let H, G be digraphs and assume that H is a minor of G. Then $\mathbf{pw}(H) \leq \mathbf{pw}(G)$.*

Proof. Let W be a path decomposition of G of width $\mathbf{pw}(G)$, and let η be a minor model of H in G. We construct a path decomposition W' of H as follows. Take the decomposition W and

- for every $u \in V(H)$, replace each occurrence of a vertex from $\eta(u)$ with u (and remove duplicates, if necessary);
- remove all the remaining vertices of G from all the bags of decomposition W.

Clearly, bags of W' contain only vertices of H, and they are of size at most $\mathbf{pw}(G)+1$, so it remains to prove that W' is a correct path decomposition of H.

Firstly, take any vertex $u \in V(H)$ and observe that the set of all the bags of W' containing u consists of bags with indices from $\bigcup_{x \in \eta(u)} I^W_x$. Clearly $\bigcup_{x \in \eta(u)} I^W_x$ is nonempty, and we claim that it is an interval. For the sake of contradiction assume that $\bigcup_{x \in \eta(u)} I^W_x$ is not an interval. Hence, $\eta(u)$ can be partitioned into two nonempty subsets R_1, R_2, such that in W every interval of a vertex of R_1 would end before the start of any interval of a vertex of R_2. However, since $\eta(u)$ is strongly connected, then there must be an arc from R_1 to R_2 in G, which is a contradiction with the fact that W is a path decomposition of G. We infer that $\bigcup_{x \in \eta(u)} I^W_x$ is a nonempty interval, so properties (i) and (ii) are satisfied.

Secondly, take any arc $(u,v) \in V(H)$. By the definition of the minor model, there exists $x \in \eta(u)$ and $y \in \eta(v)$ such that $(x,y) \in E(G)$. Then property (iii) for the arc (u,v) in decomposition W' follows from property (iii) for the arc (x,y) in decomposition W, and the fact that $I^W_x \subseteq I^{W'}_u$ and $I^W_y \subseteq I^{W'}_v$. □

Similarly as in the case of treewidth, for the sake of constructing dynamic programming routines it is convenient to work with *nice* path decompositions. We say that a path decomposition $W = (W_1, \ldots, W_r)$ is *nice* if it has following two additional properties:

- $W_1 = W_r = \emptyset$;
- for every $i = 1, 2, \ldots, r-1$ we have that $W_{i+1} = W_i \cup \{v\}$ for some vertex $v \notin W_i$, or $W_{i+1} = W_i \setminus \{w\}$ for some vertex $w \in W_i$.

If $W_{i+1} = W_i \cup \{v\}$ then we say that in bag W_{i+1} we *introduce* vertex v, while if $W_{i+1} = W_i \setminus \{w\}$ then we say that in bag W_{i+1} we *forget* vertex w. Given any path decomposition W of width p, in

$\mathcal{O}(p|V(T)|)$ time we can construct a nice path decomposition W' of the same width in a standard manner: we first introduce empty bags at the beginning and at the end, and then between any two consecutive bags W_i, W_{i+1} we insert a sequence of new bags by first forgetting the vertices of $W_i \setminus W_{i+1}$, and then introducing the vertices of $W_{i+1} \setminus W_i$.

The following lemma uses the concept of a nice path decomposition to prove existence of vertices of small out- and indegrees.

Lemma 30. *Let T be a semi-complete digraph of pathwidth at most k. Then T contains a vertex of outdegree at most k, and a vertex of indegree at most k.*

Proof. Let $W = (W_1, \ldots, W_r)$ be a nice path decomposition of T of width at most k, and let v_0 be the vertex that is forgotten first in this path decomposition, i.e., the index i of the bag where it is forgotten is minimum. By minimality of i and the definition of path decomposition, every vertex that is an outneighbor of v_0 needs to be contained in W_i. Since there is at most k vertices other than v_0 in W_i, it follows that the outdegree of v_0 is at most k. The proof for indegree is symmetric — we take v_0 to be the vertex that is introduced last. □

Path decompositions can be viewed also from a different perspective: a path decomposition naturally corresponds to a monotonic sequence of separations. This viewpoint will be very useful when designing exact and approximation algorithms for pathwidth.

Definition 31. *A sequence of separations $((A_0, B_0), \ldots, (A_r, B_r))$ is called a* separation chain *if $(A_0, B_0) = (\emptyset, V(T))$, $(A_r, B_r) = (V(T), \emptyset)$ and $A_i \subseteq A_j, B_i \supseteq B_j$ for all $i \leq j$. The* width *of the separation chain is equal to* $\max_{1 \leq i \leq r} |A_i \cap B_{i-1}| - 1$.

Lemma 32. *The following holds.*

- *Let $W = (W_1, \ldots, W_r)$ be a path decomposition of a digraph T of width at most p. Then sequence $((A_0, B_0), \ldots, (A_r, B_r))$ defined as $(A_i, B_i) = (\bigcup_{j=1}^{i} W_j, \bigcup_{j=i+1}^{r} W_j)$ is a separation chain in T of width at most p.*
- *Let $((A_0, B_0), \ldots, (A_r, B_r))$ be a separation chain in a digraph T of width at most p. Then $W = (W_1, \ldots, W_r)$ defined by $W_i = A_i \cap B_{i-1}$ is a path decomposition of T of width at most p.*

Proof. For the first claim, it suffices to observe that (A_i, B_i) are indeed separations, as otherwise there would be an edge $(v, w) \in E(T)$ such that v can belong only to bags with indices at most i and w can belong only to bags with indices larger than i; this is a contradiction with property (iii) of path decomposition. The bound on width follows from the fact that $A_i \cap B_{i-1} = W_i$ by property (ii) of path decomposition.

For the second claim, observe that the bound on width follows from the definition of a separation chain. It remains to carefully check all the properties of a path decomposition. Property (i) follows from the fact that $A_0 = \emptyset$, $A_r = V(T)$ and $W_i \supseteq A_i \setminus A_{i-1}$ for all $1 \leq i \leq r$. Property (ii) follows from the fact that $A_i \subseteq A_j, B_i \supseteq B_j$ for all $i \leq j$: the interval in the decomposition containing any vertex v corresponds to the intersection of the prefix of the chain where v belongs to sets A_i, and the suffix where v belongs to sets B_i.

For property (iii), take any $(v, w) \in E(T)$. Let α be the largest index such that $v \in W_\alpha$ and β be the smallest index such that $w \in W_\beta$. It suffices to prove that $\alpha \geq \beta$. For the sake of contradiction assume that $\alpha < \beta$ and consider separation (A_α, B_α). By maximality of α it follows that $v \notin B_\alpha$; as $\beta > \alpha$ and β is minimal, we have also that $w \notin A_\alpha$. Then $v \in A_\alpha \setminus B_\alpha$ and $w \in B_\alpha \setminus A_\alpha$, which contradicts the fact that (A_α, B_α) is a separation. □

Note that the transformations in the first and in the second claim of Lemma 32 are inverse to each other and can be carried out in $\mathcal{O}(p|V(T)|)$ assuming that we store separations along with separators. Hence, instead of looking for a path decomposition of width p one may look for a separation chain of width at most p. Note also that if decomposition W does not contain a pair of consecutive equal bags, then the corresponding separation chain has only separations of order at most p.

Assume that $((A_0, B_0), \ldots, (A_r, B_r))$ is a separation chain corresponding to a nice path decomposition W in the sense of Lemma 32. Since $A_0 = \emptyset$, $A_r = V(G)$, and $|A_i|$ can change by at most 1 between two consecutive separations, for every ℓ, $0 \leq \ell \leq |V(G)|$, there is some separation (A_i, B_i) for which $|A_i| = \ell$ holds. Let $W[\ell]$ denote any such separation; note that the order of $W[\ell]$ is at most the width of W.

Before we proceed, let us state one simple, but very important fact about separations in semi-complete digraphs. Assume that T is a semi-complete digraph and let (A, B) be a separation of T. We know that $E(A \setminus B, B \setminus A) = \emptyset$, so $E(B \setminus A, A \setminus B) = (B \setminus A) \times (A \setminus B)$ because T is semi-complete. A simple application of this observation is the following. If W is a nice path decomposition of T, then by Lemma 32 every bag W_i is a separator separating the vertices that are not yet introduced from the vertices that are already forgotten. Therefore, there is no arc from a vertex that is forgotten to a vertex that is not yet introduced, but from every vertex not yet introduced there is an arc to every vertex that is already forgotten.

Cliquewidth

Finally, we introduce the notion of cliquewidth for digraphs. The definition for digraphs is identical to the definition for undirected graphs, cf. Section 2.3.3, with exception of the join operation. In the directed setting join operation $\eta_{i,j}((G, \alpha))$ creates all possible arcs with tail labeled with i and head labeled with j. We remark that this definition appeared already in the original paper of Courcelle and Olariu [83] where the concept of cliquewidth has been introduced.

Comparison of the parameters

Lemma 33. *For every digraph D, it holds that $\mathbf{pw}(D) \leq 2 \cdot \mathbf{ctw}(D)$. Moreover, given an ordering of $V(D)$ of cutwidth c, one can in $\mathcal{O}(|V(D)|^2)$ time compute a path decomposition of width at most $2c$.*

Proof. We provide a method of construction of a path decomposition of width at most $2c$ from an ordering of $V(D)$ of cutwidth c.

Let $(v_1, v_2, \ldots, v_n)$ be the ordering of $V(D)$ of cutwidth c. Let $F \subseteq E(D)$ be the set of edges (v_j, v_i) such that $j > i$; edges from F will be called *back edges*. We now construct a path decomposition $W = (W_1, W_2, \ldots, W_n)$ of D by setting

$$W_\ell = \{v_\ell\} \cup \bigcup \{\{v_i, v_j\} \mid i \leq \ell < j \ \wedge \ (v_j, v_i) \in E(D)\}.$$

In other words, for each cut between two consecutive vertices in the order (plus one extra at the end of the ordering) we construct a bag that contains (i) endpoints of all the back edges that are cut by this cut, and (ii) the last vertex before the cut. Observe that $|W_\ell| \leq 2c+1$ for every $1 \leq \ell \leq n$. It is easy to construct W in $\mathcal{O}(|V(D)|^2)$ time using one scan through the ordering $(v_1, v_2, \ldots, v_n)$. We are left with arguing that W is a path decomposition. Clearly $\bigcup_{i=1}^{n} W_i = V(D)$, so property (i) holds

Consider any vertex v_ℓ and an index $j \neq \ell$, such that $v_\ell \in W_j$. Assume first that $j < \ell$. By the definition of W, there exists an index $i \leq j$ such that $(v_\ell, v_i) \in E(D)$. Existence of this arc

implies that v_ℓ has to be contained in every bag between W_j and W_ℓ, by the definition of W. A symmetrical reasoning works also for $j \geq \ell$. We infer that for every vertex v_ℓ the set of bags it is contained in form an interval in the path decomposition. This proves property (ii).

To finish the proof, consider any edge $(v_i, v_j) \in E(G)$. If $i > j$ then $\{v_i, v_j\} \subseteq W_j$, whereas if $i < j$ then $v_i \in W_i$ and $v_j \in W_j$. Thus, property (iii) holds as well. □

Lemma 34. *For every semi-complete digraph T, it holds that $\mathbf{cw}(T) \leq \mathbf{pw}(T) + 2$. Moreover, given a path decomposition of width p, one can in $\mathcal{O}(|V(T)|^2)$ time compute a clique expression constructing T using $p + 2$ labels.*

Proof. We provide a method of construction of a clique expression using $p + 2$ labels from a path decomposition of width p. Let $(W_1, W_2, \ldots, W_r)$ be a path decomposition of T of width p. Without loss of generality we can assume that the given path decomposition is nice. As in a nice path decomposition every vertex is introduced and forgotten exactly once, we have that $r = 2|V(T)| + 1$.

We now build a clique expression using $p + 2$ labels that constructs the semi-complete digraph T along the path decomposition. Intuitively, at each step of the construction, every vertex of the bag W_i is assigned a different label between 1 and $p + 1$, while all forgotten vertices are assigned label $p + 2$. If we proceed in this manner, we will end up with the whole digraph T labeled with $p + 2$, constructed for the last bag. As we begin with an empty graph, we just need to show what to do in the introduce and forget vertex steps.

Introduce vertex step.

Assume that $W_i = W_{i-1} \cup \{v\}$, i.e., bag W_i introduces vertex v. Note that this means that $|W_{i-1}| \leq p$. As labels from 1 up to $p + 1$ are assigned to vertices of W_{i-1} and there are at most p of them, let q be a label that is not assigned. We perform following operations; their correctness is straightforward.

- perform $\oplus\iota_q$: we introduce the new vertex with label q;
- for each $w \in W_{i-1}$ with label q', perform join $\eta_{q,q'}$ if $(v, w) \in E(D)$ and join $\eta_{q',q}$ if $(w, v) \in E(D)$;
- perform join $\eta_{q,p+2}$ since the new vertex has an outgoing arc to every forgotten vertex.

Forget vertex step.

Assume that $W_i = W_{i-1} \setminus \{w\}$, i.e., bag W_i forgets vertex w. Let $q \in \{1, 2, \ldots, p + 1\}$ be the label of w. We just perform relabel operation $\rho_{q \to p+2}$, thus moving w to forgotten vertices. □

5.3 MSO and semi-complete digraphs

Recall that we have defined $\mathbf{MSO}_1$ and $\mathbf{MSO}_2$ logic for digraphs in Section 2.3.6. $\mathbf{MSO}_1$ is Monadic Second-Order Logic with quantification over subsets of vertices but not of arcs, while in $\mathbf{MSO}_2$ we allow also quantification over arc sets.

In the undirected setting, it is widely known that model checking $\mathbf{MSO}_2$ is fixed-parameter tractable, when the parameters are the length of the formula and the treewidth of the graph; cf. Theorem 16. As far as $\mathbf{MSO}_1$ is concerned, model checking $\mathbf{MSO}_1$ is fixed-parameter tractable, when the parameters are the length of the formula and the cliquewidth of the graph; cf. Theorem 18. These results in fact hold not only for undirected graphs, but for structures with binary

relations in general, in particular for digraphs. The following result follows from the work of Courcelle, Makowsky and Rotics [82]; we remark that the original paper treats of undirected graphs, but in fact the results hold also in the directed setting (cf. [157, 160, 202]).

Theorem 35 ([82])**.** *There exists an algorithm with running time $f(||\varphi||, k) \cdot n^2$ that given an $\mathbf{MSO}_1$ formula φ checks whether φ is satisfied in a digraph G on n vertices, given together with a clique expression using at most k labels constructing it.*

Lemma 34 asserts that cliquewidth of a semi-complete digraph is bounded by its pathwidth plus 2. Moreover, the proof gives explicit construction of the corresponding expression. Hence, the following meta-theorem follows as an immediate corollary.

Theorem 36. *There exists an algorithm with running time $f(||\varphi||, p) \cdot n^2$ that given an $\mathbf{MSO}_1$ formula φ checks whether φ is satisfied in a semi-complete digraph T on n vertices, given together with a path decomposition of width p.*

We note that by pipelining Lemmas 33 and 34 one can show that an analogous result holds also for cutwidth.

It is tempting to conjecture that the tractability result for $\mathbf{MSO}_1$ and pathwidth or cutwidth could be extended also to $\mathbf{MSO}_2$, as the decompositions resemble path decompositions in the undirected setting. We use the following lemma to show that this is unfortunately not true.

Lemma 37. *There exists a constant-size $\mathbf{MSO}_2$ formula ψ over a signature enriched with three unary relations on vertices, such that checking whether ψ is satisfied in a transitive tournament with three unary relations on vertices is NP-hard.*

Proof. We provide a polynomial-time reduction from the 3-CNF-SAT problem. We are given a boolean formula φ with n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$. We are to construct a transitive tournament T with three unary relations on vertices such that φ is satisfiable if and only if ψ, the constant size $\mathbf{MSO}_2$ formula that will be constructed while describing the reduction, is true in T. Intuitively, the unary relations in T will encode the whole formula φ, while ψ is simply an $\mathbf{MSO}_2$-definable check that nondeterministically guesses the evaluation of variables and checks it.

We will use three unary relations on vertices, denoted P, B and C. The tournament consists of $m(2n+1)+1$ vertices $v_0, v_1, \ldots, v_{m(2n+1)}$, where the edge set is defined as $E(T) = \{(v_i, v_j) \mid i > j\}$. We define B (for *border*) to be true in v_i if and only if i is divisible by $2n+1$. Thus, the whole tournament is divided into m intervals between consecutive vertices satisfying B, each of size $2n$. Each of these intervals will be responsible for checking one of the clauses. The $2n$ vertices in each interval correspond to literals of variables of φ. We define P (for *parity*) to be satisfied in every second vertex of each interval, so that P is satisfied in the first vertex of each interval. The first pair of vertices corresponds to literals $x_1, \neg x_1$, the second to $x_2, \neg x_2$ etc. In the i-th interval we make the C (for *check*) relation true in vertices corresponding to literals appearing in clause C_i. This concludes the construction.

We now build the formula ψ that checks existence of an assignment satisfying φ. The correctness of the reduction will follow directly from the construction of ψ.

Firstly, we quantify existentially over a subset of edges M and subset of vertices X, which will be the set of all vertices corresponding to literals that are true in the assignment. We will construct ψ in such a manner that M will be exactly the set of arcs $\{(v_{i+2n+1}, v_i) \mid 0 \leq i \leq (m-1)(2n+1)\}$. We use M to transfer information on the assignment between consecutive intervals.

Formally, in ψ we express the following properties of X and M. Their expressibility in $\mathbf{MSO}_2$ by formulas of constant size is straightforward. We just remark that we can test whether two vertices x, y are consecutive by checking if arc (y, x) exists and that there is no z for which arcs (y, z) and (z, x) simultaneously exist; similarly, we can test whether x, y are subsequent vertices satisfying B.

(1) $(v_{2n+1}, v_0) \in M$ (note that v_0 and v_{2n+1} can be defined as the first and second vertex satisfying B).

(2) For every two pairs of consecutive vertices v_i, v_{i+1} and v_j, v_{j+1}, if $(v_j, v_i) \in M$ then $(v_{j+1}, v_{i+1}) \in M$.

(3) For every subset $N \subseteq M$, if N satisfies (1) and (2) then $N = M$.

(4) Vertices satisfying B are not in X.

(5) For every two consecutive vertices v_i, v_{i+1}, such that $v_i, v_{i+1} \notin B$, $v_i \in P$ and $v_{i+1} \notin P$, exactly one of the vertices v_i, v_{i+1} belongs to X (exactly one literal of every variable is true).

(6) For every $(v_i, v_j) \in M$, $v_i \in X$ if and only if $v_j \in X$.

(7) For every interval between two subsequent vertices satisfying B, at least one of the vertices of this interval satisfies C and belongs to X.

Properties (1), (2) and (3) assert that $M = \{(v_{i+2n+1}, v_i) \mid 0 \leq i \leq (m-1)(2n+1)\}$. Properties (4) and (5) assert that in each interval X corresponds to some valid assignment, while property (6) asserts that the assignments in all the intervals are equal. Property (7) checks whether each of the clauses is satisfied. □

The following theorem follows immediately from Lemma 37, and shows that the tractability result for $\mathbf{MSO}_1$ cannot be extended to $\mathbf{MSO}_2$.

Theorem 38. *There exists a constant-size* $\mathbf{MSO}_2$ *formula* ψ*, such that checking whether* ψ *is true in a semi-complete digraph of constant cutwidth and pathwidth is NP-hard.*

Proof. Consider the construction of Lemma 37. Observe that one can replace each vertex with a constant size strongly connected semi-complete digraph that encodes satisfaction of unary relations. The arcs between these strongly connected digraphs are defined naturally, i.e., if (v, w) was an arc, then we put an arc between every vertex of the digraph replacing v and every vertex of the digraph replacing w. In ψ we then replace the unary relations with constant size formulas testing the shape of strongly connected components, thus obtaining a constant size formula ψ' whose model checking on semi-complete digraphs of constant pathwidth and cutwidth is NP-hard. □

We remark that Courcelle et al. [82] have proven a result similar to Lemma 37 for cliques instead of transitive tournaments. More precisely, assuming a stronger assumption $\mathrm{P}_1 \neq \mathrm{NP}_1$, there is a constant-size formula ψ of $\mathbf{MSO}_2$ such that verifying satisfaction of ψ on cliques *without any unary relations* cannot be performed in polynomial time. By ignoring arc directions, the same proof as in [82] can be carried out also in our setting. Thus we can obtain a version of Theorem 38 that assumes $\mathrm{P}_1 \neq \mathrm{NP}_1$, and provides a formula ψ whose satisfaction cannot be verified in polynomial time even on transitive tournaments. Since we include Theorem 38 for demonstrative purposes only, we omit the details here.

Chapter 6

Computing width measures of semi-complete digraphs

6.1 The obstacle zoo

In this section we describe the set of obstacles used by the algorithms. We begin with *jungles*, the original obstacle introduced by Fradkin and Seymour [153], and their enhanced versions that will be used extensively in this chapter, namely *short jungles*. It appears that the enhancement enables us to construct large topological subgraph, minor or immersion models in short jungles in a greedy manner, and this observation is the key to trimming the running times for containment tests. We then recall the notion of a *triple* that is an important concept introduced by Fradkin and Seymour [153], and which we will also use for irrelevant vertex rules in Chapter 7. Finally, we continue with further obstacles that will be used in the algorithms: degree and matching tangles for pathwidth, and backward tangles for cutwidth. Each time we describe one of these obstacles, we prove two lemmas. The first asserts that existence of the structure is indeed an obstacle for having small width, while the second shows that one can constructively find an appropriate short jungle in a sufficiently large obstacle.

6.1.1 Jungles and short jungles

Definition 39. *Let T be a semi-complete digraph and k be an integer. A* k-jungle *is a set $X \subseteq V(T)$ such that* (i) $|X| = k$*;* (ii) *for every $v, w \in X$, $v \neq w$, either $(v, w) \in E(T)$ or there are k internally vertex-disjoint paths from v to w.*

It is easy to observe that existence of a $(k+1)$-jungle is an obstacle for admitting a path decomposition of width smaller than k, as in such a decomposition there would necessarily be a bag containing all the vertices of the jungle. The work of Fradkin and Seymour [153] essentially says that containing a large jungle is the only reason for not admitting a decomposition of small width: if $\mathbf{pw}(T) \geq f(k)$ for some function f (that is actually quadratic), then T must necessarily contain a k-jungle. In this chapter we strenghten this result by providing linear bounds on function f, and moreover showing that in the obtained jungle the paths between vertices may be assumed to be short, as in the definition below.

Definition 40. *Let T be a semi-complete digraph and k, d be integers. A* (k, d)-short (immersion)

jungle *is a set* $X \subseteq V(T)$ *such that* (i) $|X| \geq k$*;* (ii) *for every* $v, w \in X$*,* $v \neq w$*, there are* k *internally vertex-disjoint (edge-disjoint) paths from* v *to* w *of length at most* d*.*

We remark that in this definition we treat a path as a subdigraph. Thus, among the k vertex-disjoint paths from v to w only at most one can be of length 1. We remark also that in all our algorithms, every short jungle is constructed and stored together with corresponding families of k paths for each pair of vertices.

The restriction on the length of the paths enables us to construct topological subgraph and immersion models in short jungles greedily. This is the most useful advantage of short jungles over jungles, as it enables us to reduce the time complexity of containment tests to single-exponential.

Lemma 41. *If a digraph* T *contains a* (dk, d)*-short (immersion) jungle for some* $d > 1$*, then it admits every digraph* S *with* $|S| \leq k$ *as a topological subgraph (as an immersion).*

Proof. Firstly, we prove the lemma for short jungles and topological subgraphs. Let X be the short jungle whose existence is assumed. We construct the expansion greedily. As images of vertices of S we put arbitrary $|V(S)|$ vertices of X. Then we construct paths being images of arcs in S; during each construction we use at most $d-1$ new vertices of the digraph for the image. While constructing the i-th path, which has to lead from v to w, we consider dk vertex-disjoint paths of length d from v to w. So far we used at most $k + (i-1)(d-1) < dk$ vertices for images, so at least one of these paths does not traverse any used vertex. Hence, we can safely use this path as the image and proceed; note that thus we use at most $d-1$ new vertices.

Secondly, we prove the lemma for short immersion jungles and immersions. Let X be the short immersion jungle whose existence is assumed. We construct the immersion greedily. As images of vertices of S we put arbitrary $|V(S)|$ vertices of X. Then we construct paths being images of arcs in S; during each construction we use at most d new arcs of the digraph for the image. While constructing the i-th path, which has to lead from v to w, we consider dk edge-disjoint paths of length d from v to w. So far we used at most $(i-1)d < dk$ arcs, so at least one of these paths does not contain any used arc. Hence, we can safely use this path as the image and proceed; note that thus we use at most d new arcs. □

We now prove the analogue of Lemma 41 for minors. We remark that unlike Lemma 41, the proof of the following lemma is nontrivial.

Lemma 42. *If a digraph* T *contains a* $(2dk+1, d)$*-short jungle for some* $d > 1$*, then it admits every digraph* S *with* $|S| \leq k$ *as a minor.*

Proof. Let S' be a digraph constructed as follows: we take S and whenever for some vertices v, w arc (v, w) exists but (w, v) does not, we add also the arc (w, v). We have that $|S'| \leq 2|S| \leq 2k$, so since T contains a $(2dk+1, d)$-jungle, by Lemma 41 we infer that T contains S' as a topological subgraph. Moreover, since for every two vertices of this jungle there is at most one path of length 1 between them, in the construction of Lemma 41 we may assume that we always use one of the paths of length longer than 1. Hence, we can assume that all the paths in the constructed expansion of S' in T are of length longer than 1.

Let η' be the constructed expansion of S' in T. Basing on η', we build a minor model η of S' in T; since S is a subdigraph of S', the lemma will follow. We start with $\eta(v) = \{\eta'(v)\}$ for every $v \in V(S')$ and gradually add vertices to each $\eta(v)$.

Consider two vertices $v, w \in V(S')$ such that $(v,w),(w,v) \in E(S')$. We have then two vertex disjoint paths $P = \eta'((v,w))$ and $Q = \eta'((w,v))$ that lead from $\eta'(v)$ to $\eta'(w)$ and vice versa. We know that P and Q are of length at least 2. Moreover, without loss of generality we may assume that for any pair of vertices (x,y) on P that are (i) not consecutive, (ii) y appears on P_1 after x, and (iii) $(x,y) \neq (\eta'(v), \eta'(w))$, we have that $(y,x) \in E(T)$. Indeed, otherwise we would have that $(x,y) \in E(T)$ and path P could be shortcutted using arc (x,y) without spoiling the property that it is longer than 1. We can assume the same for the path Q.

Assume first that one of the paths P, Q is in fact of length greater than 2. Assume without loss of generality that it is P, the construction for Q is symmetric. Let p_1 and p_2 be the first and the last internal vertex of P, respectively. By our assumptions about nonexistence of shortcuts on P and about $|P| > 2$, we know that $p_1 \neq p_2$, $(p_2, \eta'(v)) \in E(T)$ and $(\eta'(w), p_1) \in E(T)$. Observe that the subpath of P from $\eta'(v)$ to p_2, closed by the arc $(p_2, \eta'(v))$ forms a directed cycle. Include the vertex set of this cycle into $\eta(v)$. Observe that thus we obtain an arc $(p_2, \eta'(w))$ from $\eta(v)$ to $\eta(w)$, and an arc $(\eta'(w), p_1)$ from $\eta(w)$ to $\eta(v)$.

Now assume that both of the paths P, Q are of length exactly 2, that is, $P = \eta'(v) \to p \to \eta'(w)$ for some vertex p, and $Q = \eta'(w) \to q \to \eta'(v)$ for some vertex $q \neq p$. Since T is semi-complete, at least one of arcs $(p,q),(q,p)$ exists. Assume without loss of generality that $(p,q) \in E(T)$; the construction in the second case is symmetric. Note that $\eta'(v) \to p \to q \to \eta'(v)$ is a directed cycle; include the vertex set of this cycle into $\eta(v)$. Observe that thus we obtain an arc $(p, \eta'(w))$ from $\eta(v)$ to $\eta(w)$, and an arc $(\eta'(w), q)$ from $\eta(w)$ to $\eta(v)$.

Concluding, for every $v \in V(T)$ the final $\eta(v)$ consists of $\eta'(v)$ plus vertex sets of directed cycles that pairwise meet only in $\eta'(v)$. Thus, $T[\eta(v)]$ is strongly connected for each $v \in V(T)$. Moreover, for every pair of vertices $v, w \in V(S')$ such that $(v,w),(w,v) \in E(S')$, we have pointed out an arc from $\eta(v)$ to $\eta(w)$ and from $\eta(w)$ to $\eta(v)$. Hence, η is a minor model of S'. □

6.1.2 Triples

We now recall the notion of a *triple*, an obstacle extensively used in the approach of Fradkin and Seymour [153].

Definition 43. *Let T be a semi-complete digraph. A triple of pairwise disjoint subsets (A,B,C) is called a k-triple if $|A| = |B| = |C| = k$ and there exist orderings $(a_1,\ldots,a_k)$, $(b_1,\ldots,b_k)$, $(c_1,\ldots,c_k)$ of A, B, C, respectively, such that for all indices $1 \leq i,j \leq k$ we have $(a_i,b_j),(b_i,c_j) \in E(T)$ and for each index $1 \leq i \leq k$ we have $(c_i,a_i) \in E(T)$.*

The main observation of [153] is that for some function f, if a semi-complete digraph contains an $f(k)$-jungle, then it contains also a k-triple [153, (2.6)]. Moreover, if it contains a k-triple, then every digraph of size at most k is topologically contained in this triple [153, (1) in the proof of (1.1)]. We remark that Fradkin and Seymour actually attribute this observation to Chudnovsky, Scott, and Seymour. The following lemma is an algorithmic version of the aforementioned observation and will be needed in our algorithms. The proof closely follows the lines of argumentation contained in [153]; we include it for the sake of completeness.

Lemma 44. *There exists an elementary function f, such that for every $k \geq 1$ and every semi-complete graph T on n vertices, given together with a $f(k)$-jungle in it, it is possible to construct a k-triple in T in time $\mathcal{O}(n^3 \log n)$.*

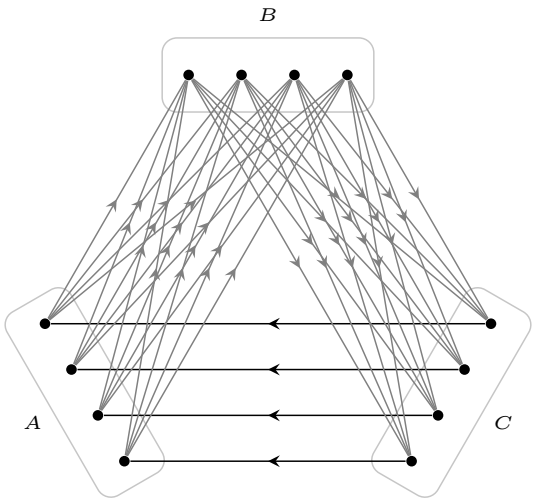

Figure 6.1: A 4-triple.

Proof. For an integer k, let $R(k,k)$ denote the Ramsey number, that is the smallest integer such that every red-blue coloring of the edges of the complete graph on $R(k,k)$ vertices contains a monochromatic clique of size k. By the theorem of Erdős and Szekeres [124], $R(k,k) \leq (1+o(1))\frac{4^{k-1}}{\sqrt{\pi k}}$. For $k \geq 1$, we define function the function f as

$$f(k) = 2^{12 \cdot 2^{R(2k,2k)}}.$$

Moreover, let $r = R(2k,2k)$, $s = 2^r$, and $m = 2^{12s} = f(k)$.

We say that a semi-complete digraph is *transitive* if it contains a transitive tournament as a subdigraph. Every tournament on m vertices contains a transitive tournament on $\log_2 m$ vertices as a subdigraph. Also an induced acyclic subdigraph of an oriented (where there is at most one directed arc between a pair of vertices) m-vertex digraph with $\log_2 m$ vertices can be found in time $\mathcal{O}(m^2 \log m)$ [274]. This algorithm can be modified into an algorithm finding a transitive semi-complete digraph in semi-complete digraphs by removing first all pairs of oppositely directed arcs, running the algorithm for oriented graphs, and then adding some of the deleted arcs to turn the acyclic digraph into a transitive semi-complete digraph. Thus, if X_0 is an m-jungle in T, then X_0 contains a subset X that is a $12s$-jungle in T and that induces a transitive semi-complete digraph. Moreover, such a set X can be found in time $\mathcal{O}(n^2 \log n)$.

The next step in the proof of Fradkin and Seymour is to partition the set X into parts X_1 and X_2 of size $6s$ each such that X_1 is complete to X_2, i.e., for each $x_1 \in X_1$ and $x_2 \in X_2$ we have $(x_1, x_2) \in E(T)$. Such a partition of the vertex set of the transitive semi-complete digraph can be easily found in time $\mathcal{O}(|X|^2)$. Because X is a $12s$-jungle in T, there are at least $6s$ vertex-disjoint paths from X_2 to X_1 in T. Indeed, otherwise by Menger's theorem there would be a separation (A,B) of order less than $6s$ such that $X_2 \subseteq A$ and $X_1 \subseteq B$, and such a separation would separate some vertex from X_1 from some vertex of X_2, contradicting existence of $6s$ internally vertex-disjoint paths between these two vertices. Let R be a minimal induced subdigraph of T such that $X \subseteq V(R)$ and there are $6s$ vertex-disjoint paths from X_2 to X_1 in R. Such a minimal subgraph R can be found in time $\mathcal{O}(n^3 \log n)$ by repeatedly removing vertices $v \in V(T) \setminus X$ if there are $6s$ vertex-disjoint paths from X_2 to X_1 in $V(T) \setminus \{v\}$. As the subroutine for finding the paths we use the

classical Ford-Fulkerson algorithm, where we finish the computation after finding $6s$ paths. Hence, the running time one application of this algorithm is $\mathcal{O}(sn^2)$. As we make at most n tests and $s = \mathcal{O}(\log n)$, the claimed bound on the runtime follows.

Let $P_1, P_2, \ldots, P_{6s}$ be vertex-disjoint paths from X_2 to X_1 in R. The arguments given by Fradkin and Seymour prove that the set of vertices Q formed by the first two and the last two vertices of these $6s$ paths contains a k-triple. Thus, by checking every triple of subsets of Q of size k in time polynomial in k, we can find a k-triple. This step takes time $\mathcal{O}\left(\binom{24s}{k}^3 k^{\mathcal{O}(1)}\right) = n^{o(1)}$, as $s = \mathcal{O}(\log n)$ and $k = \mathcal{O}(\log \log n)$. □

6.1.3 Degree tangles

The degree tangle is intuitively a concentration of vertices with very similar outdegrees. Surprisingly, a large degree tangle forms already an obstacle for admitting a path decomposition of small width. This observation is the main idea behind the degree ordering approach that we use in this chapter.

Definition 45. *Let T be a semi-complete digraph and k, ℓ be integers. A* (k, ℓ)-degree tangle *is a set $X \subseteq V(T)$ such that* (i) $|X| \geq k$; (ii) *for every $v, w \in X$ we have $|d^+(v) - d^+(w)| \leq \ell$.*

Lemma 46. *Let T be a semi-complete digraph. If T contains a $(4k+2, k)$-degree tangle X, then* $\mathbf{pw}(T) > k$.

Proof. For the sake of contradiction, assume that T admits a (nice) path decomposition W of width at most k. Let $\alpha = \min_{v \in X} d^+(v)$ and $\beta = \max_{v \in X} d^+(v)$; we know that $\beta - \alpha \leq k$. Let $(A, B) = W[\alpha]$. Recall that $(A, B) = W[\alpha]$ is any separation in the separation chain corresponding to W in the sense of Lemma 32 such that $|A| = \alpha$. We know that $|A \cap B| \leq k$ and $|A| = \alpha$.

Firstly, observe that $X \cap (A \setminus B) = \emptyset$. This follows from the fact that vertices in $A \setminus B$ can have outneighbors only in A, so their outdegrees are upper bounded by $|A| - 1 = \alpha - 1$.

Secondly, $|X \cap (A \cap B)| \leq k$ since $|A \cap B| \leq k$.

Thirdly, we claim that $|X \cap (B \setminus A)| \leq 3k + 1$. Assume otherwise. Consider subdigraph $T[X \cap (B \setminus A)]$. Since this is a subdigraph of a semi-complete digraph of pathwidth at most k, it has also pathwidth at most k. By Lemma 30 we infer that there exists a vertex $v \in X \cap (B \setminus A)$ whose indegree in $T[X \cap (B \setminus A)]$ is at most k. Since $T[X \cap (B \setminus A)]$ is semi-complete and $|X \cap (B \setminus A)| \geq 3k+2$, we infer that the outdegree of v in $T[X \cap (B \setminus A)]$ is at least $2k + 1$. As (A, B) is a separation and T is semi-complete, all the vertices of $A \setminus B$ are also outneighbors of v in T. Note that $|A \setminus B| = |A| - |A \cap B| \geq \alpha - k$. We infer that v has at least $\alpha - k + 2k + 1 = \alpha + k + 1 > \beta$ outneighbors in T, which is a contradiction with $v \in X$.

Summing up the bounds we get $4k + 2 \leq |X| \leq k + 3k + 1 = 4k + 1$, a contradiction. □

The author would like to thank Stéphan Thomassé for an excellent remark after the presentation during STACS 2013, which enabled to prove the bound $4k+2$ in Lemma 46 instead of original $5k+2$. Consequence of this better bound is approximation ratio 6 for pathwidth of a semi-complete digraph, instead of original 7.

Lemma 47. *Let T be a semi-complete digraph and let X be a $(26k, k)$-degree tangle in T. Then X contains a $(k, 3)$-short jungle which can be found in $\mathcal{O}(k^3 n^2)$ time, where $n = |V(T)|$.*

Proof. We present a proof of the existential statement. The proof can be easily turned into an algorithm finding the jungle; during the description we make remarks at the places where it may be non-trivial to observe how the algorithm should perform to achieve the promised running-time guarantee.

By possibly trimming X, assume without loss of generality that $|X| = 26k$. Take any $v, w \in X$. We either find a $(k, 3)$-short jungle in X explicitly, or find k vertex-disjoint paths from v to w of length at most 3. If for no pair v, w an explicit short jungle is found, we conclude that X is a $(k, 3)$-short jungle itself.

Let us consider four subsets of $V(T) \setminus \{v, w\}$:

- $V^{++} = (N^+(v) \cap N^+(w)) \setminus \{v, w\}$,
- $V^{+-} = (N^+(v) \cap N^-(w)) \setminus \{v, w\}$,
- $V^{-+} = (N^-(v) \cap N^+(w)) \setminus \{v, w\}$,
- $V^{--} = (N^-(v) \cap N^-(w)) \setminus \{v, w\}$.

Clearly, $|V^{++}| + |V^{-+}| + |V^{+-}| + |V^{--}| \geq n - 2$. Note that equality holds for the tournament case — in this situation these four subsets form a partition of $V(T) \setminus \{v, w\}$.

If $|V^{+-}| \geq k$, then we already have k vertex-disjoint paths of length 2 from v to w. Assume then that $|V^{+-}| < k$.

Observe that $d^+(v) \leq |V^{++}| + |V^{+-}| + 1$ and $d^+(w) \geq |V^{++}| + |V^{-+} \setminus V^{++}|$. Since $v, w \in X$, we have that $d^+(w) - d^+(v) \leq k$, so

$$|V^{-+} \setminus V^{++}| \leq d^+(w) - |V^{++}| \leq k + d^+(v) - |V^{++}| \leq k + 1 + |V^{+-}| \leq 2k.$$

Let $A = V^{++} \setminus V^{+-}$ and $B = V^{--} \setminus V^{+-}$. Note that A and B are disjoint, since $V^{++} \cap V^{--} \subseteq V^{+-}$. Let H be a bipartite graph with bipartition (A, B), such that for $a \in A$ and $b \in B$ we have $ab \in E(H)$ if and only if $(a, b) \in E(T)$. Every edge ab of H gives raise to a path $v \to a \to b \to w$ of length 3 from v to w. Hence, if we could find a matching of size k in H, then this matching would form a family of k vertex disjoint paths of length at most 3 from v to w. Note that testing existence of such a matching can be done in $\mathcal{O}(kn^2)$ time, as we can run the algorithm finding an augmenting path at most k times.

Assume then that such a matching does not exist. By Kőnig's theorem we can find a vertex cover C of H of cardinality smaller than k; again, this can be found in $\mathcal{O}(kn^2)$ time. As $A \cup B \cup (V^{-+} \setminus V^{++}) \cup V^{+-} = V(T) \setminus \{v, w\}$ while $(V^{-+} \setminus V^{++}) \cup V^{+-}$ contains at most than $3k - 1$ vertices in total, $A \cup B$ must contain at least $(26k - 2) - (3k - 1) = 23k - 1$ vertices from X. We consider two cases: either $|A \cap X| \geq 16k$, or $|B \cap X| \geq 7k$.

Case 1. In the first case, consider set $Y_0 = X \cap (A \setminus C)$. Since $|A \cap X| \geq 16k$ and $|A \cap C| < k$, we have that $|Y_0| > 15k$. Let Y be any subset of Y_0 of size $15k$. Take any vertex $y \in Y$ and consider, where its outneighbors can lie. These outneighbors can be either in $\{v\}$ (at most 1 of them), in $(V^{-+} \setminus V^{++}) \cup V^{+-}$ (less than $3k$ of them), in $B \cap C$ (at most k of them), or in A. As $d^+(v) \geq |A|$ and $v, y \in X$, we have that $d^+(y) \geq |A| - k$. We infer that y must have at least $|A| - 5k$ outneighbors in A. As $|Y| = 15k$, we have that y has at least $10k$ outneighbors in Y.

Note that in the tournament case we would be already finished, as this lower bound on the outdegree would imply also an upper bound on indegree, which would contradict the fact that $T[Y]$ contains a vertex of indegree at least $\frac{|Y|-1}{2}$. This also shows that in the tournament setting a stronger claim holds that in fact X is a $(k, 3)$-jungle itself. In the semi-complete setting, however, we do not

have any contradiction yet. In fact no contradiction is possible as the stronger claim is no longer true. To circumvent this problem, we show how to find an explicit $(k,3)$-short jungle within Y.

Observe that the sum of outdegrees in $T[Y]$ is at least $10k \cdot 15k = 150k^2$. We claim that the number of vertices in Y that have indegrees at least $6k$ is at least k. Otherwise, the sum of indegrees would be bounded by $15k \cdot k + 6k \cdot 14k = 99k^2 < 150k^2$ and the sums of the indegrees and of the outdegrees would not be equal. Let Z be any set of k vertices in Y that have indegrees at least $6k$ in $T[Y]$. Take any $z_1, z_2 \in Z$ and observe that in $T[Y]$ the set of outneighbors of z_1 and the set of inneighbors of z_2 must have intersection of size at least k, as $d^+_{T[Y]}(z_1) \geq 10k$, $d^-_{T[Y]}(z_2) \geq 6k$ and $|Y| = 15k$. Through these k vertices one can rout k vertex-disjoint paths from z_1 to z_2, each of length 2. Hence, Z is the desired $(k,3)$-short jungle.

Case 2. This case will be similar to the previous one, with the exception that we only get a contradiction: there is no subcase with finding an explicit smaller jungle. Consider set $Y_0 = X \cap (B \setminus C)$. Since $|B \cap X| \geq 7k$ and $|B \cap C| < k$, we have that $|Y_0| > 6k$. Let Y be any subset of Y_0 of size $6k+1$. Take any vertex $y \in Y$ that has outdegree at least $3k$ in $T[Y]$ (since $|Y| = 6k+1$, such a vertex exists), and consider its outneighbors. As $y \notin C$ we have that all the vertices of $A \setminus C$ are the outneighbors of y (more than $|A| - k$ of them), and there are at least $3k$ outneighbors within B. Hence $d^+(y) > |A| + 2k$. On the other hand, the outneighbors of v have to lie inside $A \cup V^{+-} \cup \{w\}$, so $d^+(v) \leq |A \cup V^{+-} \cup \{w\}| \leq |A| + k$. We infer that $d^+(y) - d^+(v) > k$, which is a contradiction with $v, y \in X$. □

6.1.4 Matching tangles

Definition 48. *Let T be a semi-complete digraph and k, ℓ be integers. A* (k,ℓ)-matching tangle *is a pair of disjoint subsets $X, Y \subseteq V(T)$ such that* (i) $|X| = |Y| = k$; (ii) *there exists a matching from X to Y, i.e., there is a bijection $f : X \to Y$ such that $(v, f(v)) \in E(T)$ for all $v \in X$;* (iii) *for every $v \in X$ and $w \in Y$ we have that $d^+(w) > d^+(v) + \ell$.*

Lemma 49. *Let T be a semi-complete digraph. If T contains a $(k+1,k)$-matching tangle (X,Y), then $\mathbf{pw}(T) > k$.*

Proof. For the sake of contradiction assume that T has a (nice) path decomposition W of width at most k. Let $\alpha = \min_{w \in Y} d^+(w)$ and let $(A,B) = W[\alpha]$. Recall that $|A| = \alpha$ and $|A \cap B| \leq k$.

Firstly, we claim that $X \subseteq A$. Assume otherwise that there exists some $v \in (B \setminus A) \cap X$. Note that all the vertices of $A \setminus B$ are outneighbors of v, so $d^+(v) \geq |A| - k = \alpha - k$. Hence $d^+(v) \geq d^+(w) - k$ for some $w \in Y$, which is a contradiction.

Secondly, we claim that $Y \subseteq B$. Assume otherwise that there exists some $w \in (A \setminus B) \cap Y$. Then all the outneighbors of w are in A, so there is less than α of them. This is a contradiction with the definition of α.

As $|A \cap B| \leq k$ and there are $k+1$ disjoint pairs of form $(v, f(v)) \in E(T)$ for $v \in X$, we conclude that there must be some $v \in X$ such that $v \in A \setminus B$ and $f(v) \in B \setminus A$. This contradicts the fact that (A,B) is a separation. □

Lemma 50. *Let T be a semi-complete digraph and let (X,Y) be a $(5k,3k)$-matching tangle in T. Then Y contains a $(k,4)$-short jungle which can be found in $\mathcal{O}(k^3 n)$ time, where $n = |V(T)|$.*

Proof. We present the proof of the existential statement; all the steps of the proof are easily constructive and can be performed within the claimed complexity bound.

Let Z be the set of vertices of Y that have indegrees at least $k+1$ in $T[Y]$. We claim that $|Z| \geq k$. Otherwise, the sum of indegrees in $T[Y]$ would be at most $k \cdot 5k + 4k \cdot k = 9k^2 < \binom{5k}{2}$, so the total sum of indegrees would be strictly smaller than the number of arcs in the digraph. It remains to prove that Z is a $(k,4)$-short jungle.

Take any $v, w \in Z$; we are to construct k vertex-disjoint paths from v to w, each of length at most 4. Let $R_0 = N^-_{T[Y]}(w) \setminus \{v\}$. Note that $|R_0| \geq k$, hence let R be any subset of R_0 of cardinality k and let $P = f^{-1}(R)$. We are to construct k vertex-disjoint paths of length 2 connecting v with every vertex of P and not passing through $P \cup R \cup \{w\}$. By concatenating these paths with arcs of the matching f between P and R and arcs leading from R to w, we obtain the family of paths we look for.

The paths from v to P are constructed in a greedy manner, one by one. Each path construction uses exactly one vertex outside $P \cup R$. Let us take the next, i-th vertex $p \in P$. As $d^+(v) > d^+(p) + 3k$, by Lemma 20 there exist at least $3k$ vertices in T that are both outneighbors of v and inneighbors of p. At most $2k$ of them can be inside $P \cup R$, at most $i - 1 \leq k - 1$ of them were used for previous paths, so there is at least one that is still unused; let us denote it by q. If in fact $q = w$, we build a path of length 1 directly from v to w thus ignoring vertex p; otherwise we can build the path of length 2 from v to p via q and proceed to the next vertex of P. □

6.1.5 Backward tangles

Definition 51. *Let T be a semi-complete digraph and k be an integer. A* k-backward tangle *is a partition (X, Y) of $V(T)$ such that* (i) *there exist at least k arcs directed from X to Y;* (ii) *for every $v \in X$ and $w \in Y$ we have that $d^+(w) \geq d^+(v)$.*

Lemma 52. *Let T be a semi-complete digraph. If T contains an $(m+1)$-backward tangle (X, Y) for $m = 64k^2 + 18k + 1$, then $\mathbf{ctw}(T) > k$.*

Proof. For the sake of contradiction, assume that $V(T)$ admits an ordering π of width at most k. Let α be the largest index such that $(X_\alpha, Y_\alpha) = (\pi[\alpha], V(T) \setminus \pi[\alpha])$ satisfies $Y \subseteq Y_\alpha$. Similarly, let β be the smallest index such that $(X_\beta, Y_\beta) = (\pi[\beta], V(T) \setminus \pi[\beta])$ satisfies $X \subseteq X_\beta$. Note that $|E(X_\alpha, Y_\alpha)|, |E(X_\beta, Y_\beta)| \leq k$. Observe also that $\alpha \leq \beta$; moreover, $\alpha < |V(T)|$ and $\beta > 0$, since X, Y are non-empty.

Let $(X_{\alpha+1}, Y_{\alpha+1}) = (\pi[\alpha+1], V(T) \setminus \pi[\alpha+1])$. By the definition of α there is a unique vertex $w \in X_{\alpha+1} \cap Y$. Take any vertex $v \in V(T)$ and suppose that $d^+(w) > d^+(v) + (k+1)$. By Lemma 20, there exist $k+1$ vertex-disjoint paths of length 2 from w to v. If v was in $Y_{\alpha+1}$, then each of these paths would contribute at least one arc to the set $E(X_{\alpha+1}, Y_{\alpha+1})$, contradicting the fact that $|E(X_{\alpha+1}, Y_{\alpha+1})| \leq k$. Hence, every such v belongs to $X_{\alpha+1}$ as well. By Lemma 46 we have that the number of vertices with outdegrees in the interval $[d^+(w) - (k+1), d^+(w)]$ is bounded by $8k+1$, as otherwise they would create a $(8k+2, 2k)$-degree tangle, implying by Lemma 46 that $\mathbf{pw}(T) > 2k$ and, consequently by Lemma 33, that $\mathbf{ctw}(T) > k$ (here note that for $k = 0$ the lemma is trivial). As $X_\alpha = X_{\alpha+1} \setminus \{w\}$ is disjoint with Y and all the vertices of X have degrees at most $d^+(w)$, we infer that $|X \setminus X_\alpha| \leq 8k+1$.

A symmetrical reasoning shows that $|Y \setminus Y_\beta| \leq 8k+1$. Now observe that

$$\begin{aligned} |E(X,Y)| &\leq |E(X_\alpha, Y)| + |E(X, Y_\beta)| + |E(X \setminus X_\alpha, Y \setminus Y_\beta)| \\ &\leq |E(X_\alpha, Y_\alpha)| + |E(X_\beta, Y_\beta)| + |E(X \setminus X_\alpha, Y \setminus Y_\beta)| \\ &\leq k + k + (8k+1)^2 = 64k^2 + 18k + 1. \end{aligned}$$

This is a contradiction with (X, Y) being an $(m+1)$-backward tangle. □

Lemma 53. *Let T be a semi-complete digraph and let (X, Y) be an m-backward tangle in T for $m = 109^2k^2$. Then X or Y contains a $(k, 4)$-short immersion jungle which can be found in $\mathcal{O}(k^3n^2)$ time, where $n = |V(T)|$.*

Proof. We present the proof of the existential statement; all the steps of the proof are easily constructive and can be performed within the claimed complexity bound.

Let $P_0 \subseteq X$ and $Q_0 \subseteq Y$ be the sets of heads and of tails of arcs from $E(X, Y)$, respectively. As $|E(X, Y)| \leq |P_0| \cdot |Q_0|$, we infer that $|P_0| \geq 109k$ or $|Q_0| \geq 109k$. Here we consider the first case; the reasoning in the second one is symmetrical.

Let P_1 be the set of vertices in P_0 that have outdegree at least $\alpha - 4k$, where $\alpha = \min_{w \in Y} d^+(w)$; note that $\alpha \geq \max_{v \in X} d^+(v)$ by the definition of a backward tangle. If there were more than $104k$ of them, they would create a $(104k, 4k)$-degree tangle, which due to Lemma 47 contains a $(4k, 3)$-short jungle, which is also a $(k, 4)$-short immersion jungle. Hence, we can assume that $|P_1| < 104k$. Let P be any subset of $P_0 \setminus P_1$ of size $5k$. We know that for any $v \in P$ and $w \in Y$ we have that $d^+(w) > d^+(v) + 4k$.

Consider semi-complete digraph $T[P]$. We have that the number of vertices with outdegrees at least k in $T[P]$ is at least k, as otherwise the sum of outdegrees in $T[P]$ would be at most $k \cdot 5k + 4k \cdot k = 9k^2 < \binom{5k}{2}$, so the sum of outdegrees would be strictly smaller than the number of arcs in the digraph. Let Z be an arbitrary set of k vertices with outdegrees at least k in $T[P]$. We prove that Z is a $(k, 4)$-short immersion jungle.

Let us take any $v, w \in Z$; we are to construct k edge-disjoint paths from v to w of length 4. Since the outdegree of v in $T[P]$ is at least k, as the first vertices on the paths we can take any k outneighbors of v in P; denote them $v_1^1, v_2^1, \ldots, v_k^1$. By the definition of P_0, each v_i^1 is incident to some arc from $E(X, Y)$. As the second vertices on the paths we choose the heads of these arcs, denote them by v_i^2, thus constructing paths $v \to v_i^1 \to v_i^2$ of length 2 for $i = 1, 2, \ldots, k$. Note that all the arcs used for constructions so far are pairwise different.

We now consecutively finish paths $v \to v_i^1 \to v_i^2$ using two more arcs in a greedy manner. Consider path $v \to v_i^1 \to v_i^2$. As $v_i^2 \in Y$ and $w \in P$, we have that $d^+(v_i^2) > d^+(w) + 4k$. Hence, by Lemma 20 we can identify $4k$ paths of length 2 leading from v_i^2 to w. At most $2k$ of them contain an arc that was used in the first phase of the construction (two first arcs of the paths), and at most $2(i-1) \leq 2k - 2$ of them can contain an arc used when finishing previous paths. This leaves us at least one path of length 2 from v_i^2 to w with no arc used so far, which we can use to finish the path $v \to v_i^1 \to v_i^2$. □

6.2 Algorithms for pathwidth

In this section we present approximation and exact algorithms for pathwidth. As described in Section 5.1.4, both of the algorithms employ the technique of sliding through the outdegree ordering of the vertices using a window of small width, and maintaining some coverage of arcs that jump over the window. This coverage can be expressed as a vertex cover of an auxiliary bipartite graph. Therefore, we start our considerations by presenting two ways of selecting a vertex cover in a bipartite graph, called *subset selectors*. The first subset selector, based on the matching theory, will be used in the approximation algorithm, while the second, based on the classical kernel for the Vertex Cover problem of Buss [54], will be used in the exact algorithm. Armed with our understanding of the introduced subset selectors, we then proceed to the description of the algorithms.

6.2.1 Subset selectors for bipartite graphs

In this subsection we propose a formalism for expressing selection of a subset of vertices of a bipartite graph. Let $\mathcal{B}$ be the class of undirected bipartite graphs with fixed bipartition, expressed as triples: left side, right side, the edge set. Let $\mu(G)$ be the size of a maximum matching in G.

Definition 54. *A function f defined on $\mathcal{B}$ is called a* subset selector *if $f(G) \subseteq V(G)$ for every $G \in \mathcal{B}$. A* reversed *subset selector f^{rev} is defined as $f^{\mathrm{rev}}((X,Y,E)) = f((Y,X,E))$. We say that subset selector f is*

- *a* vertex cover selector *if $f(G)$ is a vertex cover of G for every $G \in \mathcal{B}$, i.e., every edge of G has at least one endpoint in $f(G)$;*
- symmetric *if $f = f^{\mathrm{rev}}$;*
- monotonic *if for every graph $G = (X,Y,E)$ and its subgraph $G' = G \setminus w$ where $w \in Y$, we have that $f(G) \cap X \supseteq f(G') \cap X$ and $f(G) \cap (Y \setminus \{w\}) \subseteq f(G') \cap Y$.*

For our purposes, the goal is to find a vertex cover selector that is at the same time reasonably small in terms of $\mu(G)$, but also monotonic. As will become clear in Section 6.2.2, only monotonic vertex cover selectors will be useful for us, because monotonicity is essential for obtaining a correct path decomposition where for every vertex the set of bags containing it forms an interval. The following observation expresses, how monotonic subset selectors behave with respect to modifications of the graph; a reader well-familiar with constructing various graph decompositions will probably already see from its statement why monotonicity is so important for us. By addition of a vertex we mean adding a new vertex to the vertex set, together with an arbitrary set of edges connecting it to the old ones.

Lemma 55. *Assume that f and f^{rev} are monotonic subset selector and let $G = (X,Y,E)$ be a bipartite graph.*

- *If $v \in f(G) \cap Y$ then v stays chosen by f after any sequence of additions of vertices to the left side and deletions of vertices (different from v) from the right side.*
- *If $v \in X \setminus f(G)$ then v stays not chosen by f after any sequence of additions of vertices to the left side and deletions of vertices from the right side.*

Proof. For both claims, staying (not) chosen after a deletion on the right side follows directly from the definition of monotonicity of f. Staying (not) chosen after an addition on the left side follows from considering deletion of the newly introduced vertex and monotonicity of f^{rev}. □

The matching selector

In this section we define the subset selector that will be used for the approximation algorithm for pathwidth. Throughout this section we assume reader's knowledge of basic concepts and definitions from the classical matching theory (see [112] or [270] for reference). However, we remind the most important facts in the beginning in order to establish notation.

For a bipartite graph $G = (X,Y,E)$ with a matching M, we say that a walk W is an *alternating walk for M*, if

- W starts in a vertex that is unmatched in M;

- edges from M and outside M appear on W alternately, beginning with an edge outside M.

Whenever M is clear from the context, we omit it. If W is in fact a simple path, we say that W is an *alternating path*. A simple shortcutting arguments show that every alternating walk from v to w has an alternating path from v to w as a subsequence. Thus for every vertex $v \notin V(M)$, the set of vertices reachable by alternating walks from v is the same as the set of vertices reachable by alternating paths from v.

Assume we are given a matching M and an alternating path W with respect to M, such that either W is of even length, or the last vertex of W is unmatched in M. Then we may construct a matching M' by removing from M all the edges belonging to $E(W) \cap M$, and adding all the remaining edges of W. This operation will be called *switching along path* W. Note that if W is of even length, then $|M'| = |M|$, while if W is of odd length and the last vertex of W is unmatched, then $|M'| = |M|+1$. In the latter case we say that W is an *augmenting path for* M, since switching along W increases the size of the matching. The core observation of the classical algorithm for maximum matching in bipartite graphs is that a matching is not maximum if and only if there is an augmenting path for it.

A vertex cover of a graph is a set of vertices X such that every edge of the graph is incident to at least one vertex of X. Clearly, the size of a maximum matching is a lower bound for the size of a minimum vertex cover, as each vertex from the vertex cover can cover at most one edge of the matching. The classical Kőnig's theorem [215] states that for bipartite graphs equality holds: there exists a vertex cover of size $\mu(G)$, which is the minimum possible size. The set of arguments contained in this section in fact prove Kőnig's theorem.

The subset selector that will be used for the approximation of pathwidth is the following:

Definition 56. *By* matching selector $\mathfrak{M}$ *we denote a subset selector that assigns to every bipartite graph G the set of all the vertices of G that are matched in* **every** *maximum matching in G.*

Clearly, for any bipartite graph $G = (X, Y, E)$ we have that $|\mathfrak{M}(G) \cap X|, |\mathfrak{M}(G) \cap Y| \leq \mu(G)$, as any maximum matching of G is of size $\mu(G)$. It appears that $\mathfrak{M}$ is a symmetric and monotonic vertex cover selector. The symmetry is obvious. The crucial property of $\mathfrak{M}$ is monotonicity: its proof requires technical and careful analysis of alternating and augmenting paths in bipartite graphs. $\mathfrak{M}$ admits also an alternative characterization, expressed in Lemma 59, point (ii): it can be computed directly from any maximum matching by considering alternating paths originating in unmatched vertices. This observation can be utilized to construct an algorithm that maintains $\mathfrak{M}(G)$ efficiently during graph modifications. Moreover, from this alternative characterization it is clear that $\mathfrak{M}$ is a vertex cover selector. The following lemma expresses all the vital properties of $\mathfrak{M}$ that will be used in the approximation algorithm for pathwidth.

Lemma 57. *$\mathfrak{M}$ is a symmetric, monotonic vertex cover selector, which can be maintained together with a maximum matching of G with updates times $\mathcal{O}((\mu(G)+1) \cdot n)$ during vertex additions and deletions, where $n = |V(G)|$. Moreover, $|\mathfrak{M}(G) \cap X|, |\mathfrak{M}(G) \cap Y| \leq \mu(G)$ for every bipartite graph $G = (X, Y, E)$.*

We now proceed to the proof of Lemma 57. We split the proof into several lemmas. First, we show that $\mathfrak{M}$ is indeed monotonic.

Lemma 58. *$\mathfrak{M}$ is monotonic.*

Proof. Let $G' = G \setminus w$, where $G = (X, Y, E)$ is a bipartite graph and $w \in Y$.

Firstly, we prove that $\mathfrak{M}(G) \cap (Y \setminus \{w\}) \subseteq \mathfrak{M}(G') \cap Y$. For the sake of contradiction, assume that there is some $v \in \mathfrak{M}(G) \cap Y, v \neq w$, such that $v \notin \mathfrak{M}(G') \cap Y$. So there exists a maximum matching N of G' in which v is unmatched; we will also consider N as a (possibly not maximum) matching in G. Let M be any maximum matching in G. Since $v \in \mathfrak{M}(G)$, we have that v is matched in M. Construct a maximum path P in G that begins in v and alternates between matchings M and N (i.e., edges of P belong alternately to M and to N), starting with M. Note that every vertex of P that belongs to X is entered via an edge from M, and every vertex of P that belongs to Y is entered via an edge from N. As both w and v belong to Y and are not matched in N, this path does not enter w or v, hence it ends in some other vertex. Note also that P has length at least one as v is matched in M. Let $x \notin \{v, w\}$ be the last vertex of P. We consider two cases.

Assume first that $x \in X$, i.e., x is a vertex of the left side that is not matched in N. Then P is an augmenting path for N fully contained in G'. This contradicts the maximality of N.

Assume now that $x \in Y$, i.e., x is a vertex of the right side that is not matched in M. Then P is an alternating path for M in G and switching along P leaves v unmatched. This contradicts the fact that $v \in \mathfrak{M}(G)$.

Now we prove that $\mathfrak{M}(G) \cap X \supseteq \mathfrak{M}(G') \cap X$. We consider two cases.

Assume first that $w \in \mathfrak{M}(G)$. As w is matched in every maximum matching of G, after deleting w the size of the maximum matching drops by one: if there was a matching of the same size in G', it would constitute also a maximum matching in G that does not match w. Hence, if we take any maximum matching M of G and delete the edge incident to w, we obtain a maximum matching of G'. We infer that $\mathfrak{M}(G) \cap X \supseteq \mathfrak{M}(G') \cap X$, as every vertex of X that is unmatched in some maximum matching M in G, is also unmatched in some maximum matching G', namely in M with the edge incident to w removed.

Assume now that $w \notin \mathfrak{M}(G)$. Let M be any maximum matching of G in which w is unmatched. As M is also a matching in G', we infer that M is also a maximum matching in G' and the sizes of maximum matchings in G and G' are equal. Take any $v \in \mathfrak{M}(G') \cap X$ and for the sake of contradiction assume that $v \notin \mathfrak{M}(G)$. Let N be any maximum matching in G in which v is unmatched. Note that since $v \in \mathfrak{M}(G') \cap X$ and M is a maximum matching in G', then v is matched in M. Similarly as before, let us now construct a maximum path P in G that begins in v and alternates between matchings M and N, starting with M. Note that every vertex of P that belongs to X is entered via an edge from N, and every vertex of P that belongs to Y is entered via an edge from M. As $v \in X$, $w \in Y$, v is unmatched in N and w is unmatched in M, this path does not enter w or v, hence it ends in some other vertex. Note also that P has length at least one as v is matched in M. Let $x \notin \{v, w\}$ be the last vertex of P. Again, we consider two subcases.

In the first subcase we have $x \in X$, i.e., x is a vertex of the left side that is not matched in M. Then P is an alternating path for M in G' and switching along P leaves v unmatched. This contradicts the fact that $v \in \mathfrak{M}(G')$.

In the second subcase we have $x \in Y$, i.e., x is a vertex of the right side that is not matched in N. Then P is an augmenting path for N in G, which contradicts the maximality of N. □

In order to prove that $\mathfrak{M}$ is a vertex cover selector and can be computed efficiently, we prove the following alternative characterization. The following lemma might be considered a folklore corollary of the classical matching theory, but for the sake of completeness we include its proof.

Lemma 59. *Let $G = (X, Y, E)$ be a bipartite graph and let M be any maximum matching in G. Let $A_0 = X \setminus V(M)$ be the set of unmatched vertices on the left side, and $B_0 = Y \setminus V(M)$ be the*

set of unmatched vertices on the right side. Moreover, let A be the set of vertices of X that can be reached via an alternating path from A_0, and symmetrically let B be the set of vertices of Y reachable by an alternating path from B_0. Then

(i) there is no edge between A and B, and

(ii) $\mathfrak{M}(G) = V(G) \setminus (A \cup B)$.

Proof. We first prove point (i). For the sake of contradiction, assume that there is an edge between A and B, i.e., there is a vertex w in the set $N(A) \cap B$. Let v be a neighbor of w in A. We claim that w is reachable from A_0 via an alternating path. Assume otherwise, and take an alternating path P from A_0 reaching v. Observe that this path does not traverse w because then w would be reachable from A_0 via an alternating path. Moreover, $vw \notin M$, since if this was the case then vw would be used in P to access v. Hence, we can prolong P by the edge vw, thus reaching w by an alternation path from A_0, a contradiction. By the definition of B, w is also reachable from B_0 via an alternating path. The concatenation of these two paths is an alternating walk from A_0 to B_0, which contains an alternating simple subpath from A_0 to B_0. This subpath is an augmenting path for M, which contradicts maximality of M. Thus we infer that there is no edge between A and B, and point (i) is proven.

We now proceed to the proof that $\mathfrak{M}(G) = V(G) \setminus (A \cup B)$. On one hand, every vertex v belonging to A or B is not matched in some maximum matching: we just modify M by switching along the alternating path connecting a vertex unmatched in M with v, obtaining another maximum matching in which v is unmatched. Hence, $\mathfrak{M}(G) \subseteq V(G) \setminus (A \cup B)$. We are left with proving that $\mathfrak{M}(G) \supseteq V(G) \setminus (A \cup B)$.

Consider the set A and the set $N(A)$. We already know that $N(A)$ is disjoint with B, so also from B_0. Hence, every vertex of $N(A)$ must be matched in M. Moreover, we have that every vertex of $N(A)$ must be in fact matched to a vertex of A, as the vertices matched to $N(A)$ are also reachable via alternating walks from A_0. Similarly, every vertex of $N(B)$ is matched to a vertex of B.

We now claim that $|X \setminus A| + |N(A)| = |M|$. As $A_0 \subseteq A$, every vertex of $X \setminus A$ as well as every vertex of $N(A)$ is matched in M. Moreover, as there is no edge between A and $Y \setminus N(A)$, every edge of M has an endpoint either in $X \setminus A$ or in $N(A)$. It remains to show that no edge of M can have one endpoint in $X \setminus A$ and second in $N(A)$; this, however follows from the fact that vertices of $N(A)$ are matched to vertices of A, proved in the previous paragraph. Similarly we have that $|Y \setminus B| + |N(B)| = |M|$.

It follows that sets $(X \setminus A) \cup N(A)$ and $N(B) \cup (X \setminus B)$ are vertex covers of G of size $|M|$. Note that every vertex cover C of G of size $|M|$ must be fully matched by any matching N of size $|M|$, as every edge of N is incident to at least one vertex from C. Hence, every vertex of both $(X \setminus A) \cup N(A)$ and of $N(B) \cup (X \setminus B)$ is matched in every maximum matching of G. Since $N(A) \subseteq Y \setminus B$ and $N(B) \subseteq X \setminus A$, we have that $(X \setminus A) \cup N(A) \cup N(B) \cup (Y \setminus B) = V(G) \setminus (A \cup B)$. Thus $\mathfrak{M}(G) \supseteq V(G) \setminus (A \cup B)$. □

From now on we use the terminology introduced in Lemma 59. Note that Lemma 59 implies that sets A, B do not depend on the choice of matching M, but only on graph G. Also, point (i) of Lemma 59 shows that $\mathfrak{M}$ is a vertex cover selector. We now present an incremental algorithm that maintains $\mathfrak{M}(G)$ together with a maximum matching of G during vertex additions and deletions.

Lemma 60. *There exists an algorithm that maintains a maximum matching M and $\mathfrak{M}(G)$ for a bipartite graph $G = (X, Y, E)$ during vertex addition and deletion operations. The update time is $\mathcal{O}((\mu(G) + 1) \cdot n)$, where $n = |V(G)|$.*

Proof. Observe that, by Lemma 59, $\mathfrak{M}(G)$ can be computed from M in $\mathcal{O}(|G|)$ time: we simply compute sets A_0, B_0 and apply breadth-first search from the whole A_0 to compute A, and breadth-first search from the whole B_0 to compute B. Both of these searches take $\mathcal{O}(|G|)$ time; note that by Kőnig's theorem a bipartite graph G on n vertices can have at most $\mathcal{O}(\mu(G) \cdot n)$ edges, so $|G| = \mathcal{O}((\mu(G)+1) \cdot n)$. Hence, we just need to maintain a maximum matching.

During vertex addition, the size of the maximum matching can increase by at most 1, so we may simply add the new vertex and run one iteration of the standard breadth-first search procedure checking, whether there is an augmenting path in the new graph; this takes $\mathcal{O}(|G|)$ time. If this is the case, we modify the matching along this path and we know that we obtained a maximum matching in the new graph. Otherwise, the size of the maximum matching does not increase, so we do not need to modify the current one. Similarly, during vertex deletion we simply delete the vertex together with possibly at most one edge of the matching incident to it. The size of the stored matching might have decreased by 1; in this case again we run one iteration of checking whether there is an augmenting path, again in $\mathcal{O}(|G|)$ time. If this is the case, we augment the matching using this path, obtaining a new matching about which we know that it is maximum. Otherwise, we know that the current matching is maximum, so we do not need to modify it. □

Lemmas 58, 59 and 60 together with previous observations prove Lemma 57.

Let us conclude with the following remark. Instead of selector $\mathfrak{M}$ one could introduce a selector $\mathfrak{M}'$ defined as follows in terms of Lemma 59: $\mathfrak{M}'((X,Y,E)) = (X \setminus A) \cup N(A)$. The proof of Lemma 59 shows that this definition does not depend on the choice of maximum matching M, and moreover that selector $\mathfrak{M}'$ is in fact a vertex cover selector of size exactly $\mu(G)$. Obviously, $\mathfrak{M}'$ can be maintained in the same manner as $\mathfrak{M}$ during vertex additions and deletions, so the only missing piece is showing that both $\mathfrak{M}'$ and $\mathfrak{M}'^{\text{rev}}$ are monotonic; note here that since $\mathfrak{M}$ is symmetric, when working with $\mathfrak{M}$ we needed to perform just one such check. This claim appears to be true; however, the proof is significantly longer and more technical than the proof of Lemma 58. In addition, even though selector $\mathfrak{M}'$ has better guarantees on the size than $\mathfrak{M}$, unfortunately this gain appears to be not useful at the point when we apply $\mathfrak{M}$ in the approximation algorithm for pathwidth. In other words, replacing $\mathfrak{M}$ with $\mathfrak{M}'$ does not result in better approximation ratio, even though it may be shown that when $\mathfrak{M}'$ is applied instead of $\mathfrak{M}$, we have a better guarantee of $5k$ instead of $6k$ on the sizes of intersections of each two consecutive bags (so-called *adhesions*). Therefore, for the sake of simpler and more concise arguments we have chosen to include analysis using selector $\mathfrak{M}$ instead of $\mathfrak{M}'$.

The Buss selector

In this subsection we introduce the subset selector that will be used in the exact algorithm for pathwidth. This selector is inspired by the classical kernelization algorithm for the Vertex Cover problem of Buss [54].

Definition 61. *Let $G = (X,Y,E)$ be a bipartite graph and ℓ be a nonnegative integer. A vertex v is called* ℓ-important *if $d(v) > \ell$, and* ℓ-unimportant *otherwise. A* Buss selector *is a subset selector $\mathfrak{B}_\ell$ that returns all vertices of X that are either ℓ-important, or have at least one ℓ-unimportant neighbor.*

Note that Buss selector is highly non-symmetric, as it chooses vertices only from the left side. Moreover, it is not necessarily a vertex cover selector. However, both $\mathfrak{B}_\ell$ and $\mathfrak{B}_\ell^{\text{rev}}$ behave in a nice manner.

Lemma 62. *Both $\mathfrak{B}_\ell$ and $\mathfrak{B}_\ell^{\mathrm{rev}}$ are monotonic.*

Proof. Monotonicity of $\mathfrak{B}_\ell$ is equivalent to the observation that if $v \in X$ is ℓ-unimportant and has only ℓ-important neighbors, then deletion of any vertex of Y cannot make v ℓ-important or create ℓ-unimportant neighbors of v. This holds because the degrees of surviving neighbors of v do not change.

Monotonicity of $\mathfrak{B}_\ell^{\mathrm{rev}}$ is equivalent to the observation that if $w \in Y$ is chosen by $\mathfrak{B}_\ell^{\mathrm{rev}}$ because it is ℓ-important, then after deletion of any other $w' \in Y$ its degree does not change so it stays ℓ-important, and if it had an ℓ-unimportant neighbor, then after deletion of any other $w' \in Y$ this neighbor will still be ℓ-unimportant. □

We now prove that $\mathfrak{B}_\ell$ does not choose too many vertices unless the bipartite graph G contains a large matching.

Lemma 63. *If $|\mathfrak{B}_\ell(G)| > \ell^2 + \ell$, then G contains a matching of size $\ell + 1$.*

Proof. As $|\mathfrak{B}_\ell(G)| > \ell^2 + \ell$, in $\mathfrak{B}_\ell(G)$ there are at least $\ell + 1$ ℓ-important vertices, or at least $\ell^2 + 1$ vertices with an ℓ-unimportant neighbor. In both cases we construct the matching greedily.

In the first case we iteratively take an ℓ-important vertex of $\mathfrak{B}_\ell(G)$ and match it with any its neighbor that is not matched so far. As there is at least $\ell + 1$ of these neighbors and at most ℓ were used so far, we can always find one not matched so far.

In the second case we take an arbitrary vertex v_1 of $\mathfrak{B}_\ell(G)$ that has an ℓ-unimportant neighbor, and find any its ℓ-unimportant neighbor w_1. We add v_1w_1 to the constructed matching and mark all the at most ℓ neighbors of w_1 as used. Then we take an arbitrary unused vertex v_2 of $\mathfrak{B}_\ell(G)$ that has an ℓ-unimportant neighbor, find any its ℓ-unimportant neighbor w_2 (note that $w_2 \neq w_1$ since v_2 was not marked), add v_2w_2 to the constructed matching and mark all the at most ℓ neighbors of w_2 as used. We continue in this manner up to the point when a matching of size $\ell+1$ is constructed. Note that there will always be an unmarked vertex of $\mathfrak{B}_\ell(G)$ with an ℓ-unimportant neighbor, as at the beginning there are at least ℓ^2+1 of them and after i iterations at most $i \cdot \ell$ are marked as used. □

We prove that $\mathfrak{B}_\ell$ can be also evaluated efficiently.

Lemma 64. *There exists an algorithm which maintains $\mathfrak{B}_\ell(G)$ for a bipartite graph $G = (X, Y, E)$ during operations of vertex addition and vertex deletion with update times $\mathcal{O}(\ell n)$, where $n = |V(G)|$.*

Proof. With every vertex of X we maintain its degree and a counter of ℓ-unimportant neighbors. We also maintain degrees of vertices of Y. The degree gives us information whether the vertex is ℓ-important.

Let us first examine adding vertex v to the left side. We need to increase the degrees of neighbors of v, so some of them may become ℓ-important. For every such vertex that became ℓ-important — note that its degree is exactly $\ell + 1$ — we examine its $\ell + 1$ neighbors and decrement their ℓ-unimportant neighbors' counters. If it drops to zero, we delete this vertex from $\mathfrak{B}_\ell(G)$ unless it's ℓ-important. Finally, we count how many neighbors of v are ℓ-unimportant, set the counter appropriately and add v to $\mathfrak{B}_\ell(G)$ if necessary.

Let us now examine adding vertex w to the right side. We need to increase the degrees of neighbors of w, so some of them may become ℓ-important and thus chosen by $\mathfrak{B}_\ell(G)$. Moreover, if w is ℓ-unimportant, then we increment the ℓ-unimportant neighbors' counters of neighbors of w; if it is incremented from 0 to 1, then the vertex becomes chosen by $\mathfrak{B}_\ell(G)$, assuming that it was not already ℓ-important.

Now we examine deleting vertex v from the left side. We iterate through the neighbors of v and decrement their degrees. If any of them ceases to be ℓ-important, we iterate through all its ℓ neighbors and increment the counters of ℓ-unimportant neighbors. If for some of these neighbors the counter was incremented from 0 to 1, the neighbor becomes chosen by $\mathfrak{B}_\ell(G)$, assuming that it was not already ℓ-important.

Finally, we examine deleting vertex w from the right side. Firstly, we check if w was ℓ-important. Then we iterate through neighbors of w decreasing the degree and ℓ-unimportant neighbors' counters, if necessary. If any of the neighbors ceases to be ℓ-important or have an ℓ-unimportant neighbor, it becomes not chosen to $\mathfrak{B}_\ell(G)$. □

6.2.2 The algorithms

In this subsection we present the algorithms for computing pathwidth. We begin with the approximation algorithm and then proceed to the exact algorithm. We introduce the approximation algorithm with an additional parameter ℓ; taking $\ell = 4k$ gives the promised 6-approximation, but as we will see in Chapter 7, modifying ℓ may be useful to improve the quality of obtained degree tangle.

Theorem 65. *There exists an algorithm that given a semi-complete digraph T on n vertices and integers k and $\ell \geq 4k$, in time $\mathcal{O}(kn^2)$ outputs one of the following:*

- *an $(\ell+2, k)$-degree tangle in T;*
- *a $(k+1, k)$-matching tangle in T;*
- *a path decomposition of T of width at most $\ell + 2k$.*

In the first two cases the algorithm can correctly conclude that $\mathbf{pw}(T) > k$.

Proof. The last sentence follows from Lemmas 46 and 49. We proceed to the algorithm.

The algorithm first computes any outdegree ordering $\sigma = (v_1, v_2, \ldots, v_n)$ of $V(T)$ in $\mathcal{O}(n^2)$ time. Then in $\mathcal{O}(n)$ time we check if there is an index i such that $d^+(v_{i+\ell+1}) \leq d^+(v_i) + k$. If this is true, then $\{v_i, v_{i+1}, \ldots, v_{i+\ell+1}\}$ is an $(\ell+2, k)$-degree tangle which can be safely output by the algorithm. From now on we assume that such a situation does not occur, i.e, $d^+(v_{i+\ell+1}) > d^+(v_i) + k$ for every index i.

Let separation sequence $R_0 = ((A_0, B_0), (A_1, B_1), \ldots, (A_{n-\ell}, B_{n-\ell}))$ be defined as follows. Let us define $S_i^0 = \{v_{i+1}, v_{i+2}, \ldots, v_{i+\ell}\}$ and let $H_i = (X_i, Y_i, E_i)$ be a bipartite graph, where $X_i = \{v_1, \ldots, v_i\}$, $Y_i = \{v_{i+\ell+1}, v_{i+\ell+2}, \ldots, v_n\}$ and $xy \in E_i$ if and only if $(x, y) \in E(T)$. If $\mu(H_i) > k$, then vertices matched in a maximum matching of H_i form a $(k+1, k)$-matching tangle in T, which can be safely output by the algorithm. Otherwise, let $S_i = S_i^0 \cup \mathfrak{M}(H_i)$ and we set $A_i = X_i \cup S_i$ and $B_i = Y_i \cup S_i$; the fact that (A_i, B_i) is a separation follows from the fact that $\mathfrak{M}$ is a vertex cover selector. Finally, we add separations $(\emptyset, V(T))$ and $(V(T), \emptyset)$ at the ends of the sequence, thus obtaining separation sequence R. We claim that R is a separation chain. Note that if we prove it, by Lemma 32 the width of the corresponding path decomposition is upper bounded by $\max_{0 \leq i \leq n-\ell-1} |\{v_{i+1}, v_{i+2}, \ldots, v_{i+\ell+1}\} \cup (\mathfrak{M}(H_i) \cap X_i) \cup (\mathfrak{M}(H_{i+1}) \cap Y_{i+1})| - 1 \leq \ell + 1 + 2k - 1 = \ell + 2k$, by monotonicity of $\mathfrak{M}$ (Lemma 58).

It suffices to show that for every i we have that $A_i \subseteq A_{i+1}$ and $B_i \supseteq B_{i+1}$. This, however, follows from Lemma 55 and the fact that $\mathfrak{M}$ is symmetric and monotonic. H_{i+1} differs from H_i by deletion of one vertex on the right side and addition of one vertex on the left side, so we have

that A_{i+1} differs from A_i only by possibly incorporating vertex $v_{i+\ell+1}$ and some vertices from Y_{i+1} that became chosen by $\mathfrak{M}$, and B_{i+1} differs from B_i only by possibly losing vertex v_{i+1} and some vertices from X_i that ceased to be chosen by $\mathfrak{M}$.

Separation chain R can be computed in $\mathcal{O}(kn^2)$ time: we consider consecutive sets S_i^0 and maintain the graph H_i together with a maximum matching in it and $\mathfrak{M}(H_i)$. As going to the next set S_i^0 can be modeled by one vertex deletion and one vertex additions in graph H_i, by Lemma 57 we have that the time needed for an update is $\mathcal{O}(kn)$; note that whenever the size of the maximum matching exceeds k, we terminate the algorithm by outputting the obtained matching tangle. As we make $\mathcal{O}(n)$ updates, the time bound follows. Translating a separation chain into a path decomposition can be done in $\mathcal{O}(\ell n)$ time, since we can store the separators along with the separations when constructing them. □

We now present the exact algorithm for pathwidth.

Theorem 66. *There exists an algorithm that, given a semi-complete digraph T on n vertices and an integer k, in $2^{\mathcal{O}(k \log k)} \cdot n^2$ time computes a path decomposition of T of width at most k, or correctly concludes that no such exists.*

Proof. We say that a separation chain R in T is *feasible* if it has width at most k. By the remarks after Lemma 32, instead of looking for a path decomposition of width k we may look for a feasible separation chain. Transformation of this separation chain into a path decomposition can be carried out in $\mathcal{O}(kn)$ time, assuming that we store the separators along with the separations.

The opening step of the algorithm is fixing some outdegree ordering $\sigma = (v_1, v_2, \ldots, v_n)$ of $V(T)$. For $i = 0, 1, \ldots, n$, let H_i be a bipartite graph with left side $X_i = \{v_1, v_2, \ldots, v_i\}$ and right $Y_i = \{v_{i+1}, v_{i+2}, \ldots, v_n\}$, where $xy \in E(H_i)$ if $(x, y) \in E(T)$. As in the proof of Theorem 65, in $\mathcal{O}(n)$ time we check if there is an index i such that $d^+(v_{i+4k+1}) \leq d^+(v_i) + k$. If this is true, then $\{v_i, v_{i+1}, \ldots, v_{i+4k+1}\}$ is a $(4k+2, k)$-degree tangle and the algorithm may safely provide a negative answer by Lemma 46. From now on we assume that such a situation does not occur.

We define a subclass of *trimmed* separation chains. Every feasible separation chain can be adjusted to a trimmed separation chain by deleting some vertices from sets A_i, B_i; note that the new separation chain created in such a manner will also be feasible, as bags of the corresponding path decomposition can only get smaller. Hence, we may safely look for a separation chain that is trimmed.

Before proceeding with formal arguments, let us explain some intuition behind trimmed separation chains. A priori, a bag of a path decomposition of T of width at most k can contain some redundant vertices that do not really contribute to the separation property (Property (iii) of Definition 27). For instance, if T admits a decomposition of width $k/2$, then one could pick an arbitrary set of $k/2$ vertices of T and incorporate them in every bag; this gives us roughly $n^{k/2}$ possible decompositions, which is too large a number for an FPT algorithm. Hence, we need to restrict our attention to path decompositions that are cleaned from redundant parts of bags; these decompositions are precisely the ones that correspond to trimmed separation chains. The Buss selector $\mathfrak{B}_\ell$ will be the tool responsible for identifying which part of the bags are redundant and can be removed.

We proceed to the definition of a trimmed separation chain. We fix $m = 5k+1$. Let (A, B) be any separation in T, and let α, β be any indices between 0 and n such that $\alpha = \max(0, \beta - (4k+1))$. We say that (A, B) is *trimmed with respect to* (α, β) if (i) $Y_\beta \subseteq B \subseteq Y_\alpha \cup \mathfrak{B}_m(H_\alpha)$ and (ii) $X_\alpha \subseteq A \subseteq X_\beta \cup \mathfrak{B}_m^{\mathrm{rev}}(H_\beta)$. (A, B) is *trimmed* if it is trimmed with respect to any such pair (α, β). A separation chain is *trimmed* if every its separation is trimmed.

For a separation (A, B) let us define the *canonical index* $\beta = \beta((A, B))$ as the only integer between 0 and n such that $d^+(v_j) < |A|$ for $j \leq \beta$ and $d^+(v_j) \geq |A|$ for $j > \beta$. Similarly, the *canonical index* $\alpha = \alpha((A, B))$ is defined as $\alpha((A, B)) = \max(0, \beta((A, B)) - (4k + 1))$. Observe that by the assumed properties of ordering σ we have that vertices in X_α have outdegrees smaller than $|A| - k$.

Firstly, we observe that if (A, B) is a separation and α, β are its canonical indices, then $X_\alpha \subseteq A$ and $Y_\beta \subseteq B$. This follows from the fact that vertices in $A \setminus B$ have outdegrees smaller than $|A|$, hence they cannot be contained in Y_β, while vertices in $B \setminus A$ have outdegrees at least $|A \setminus B| \geq |A| - k$, hence they cannot be contained in X_α. Concluding, vertices of X_α may belong only to $A \setminus B$ or $A \cap B$ (left side or the separator), vertices of Y_β may belong only to $B \setminus A$ or $A \cap B$ (right side or the separator), while for vertices of the remaining part $Y_\alpha \cap X_\beta$ neither of the three possibilities is excluded.

We now show how to transform any separation chain into a trimmed one. Take any separation chain R that contains only separations of order at most k, and obtain a sequence of separations R' as follows. We take every separation (A, B) from R; let α, β be its canonical indices. We delete $X_\alpha \setminus \mathfrak{B}_m(H_\alpha)$ from B and $Y_\beta \setminus \mathfrak{B}^{\mathrm{rev}}_m(H_\beta)$ from A, thus obtaining a new pair (A', B') that is inserted into R' in place of (A, B).

Claim 1. *R' is a trimmed separation chain.*

Proof. We need to prove that every such pair (A', B') is a separation, and that all these separations form a separation chain. The fact that such a separation chain is trimmed follows directly from the definition of the performed operation and the observations on canonical indices.

First, we check that $A' \cup B' = V(T)$. This follows from the fact from A we remove only vertices of Y_β while from B we remove only vertices from X_α, but after the removal X_α is still covered by A' and Y_β by B'.

Now we check that $E(A' \setminus B', B' \setminus A') = \emptyset$. Assume otherwise, that there is a pair $(v, w) \in E(T)$ such that $v \in A' \setminus B'$ and $w \in B' \setminus A'$. By the construction of (A', B') and the fact that (A, B) was a separation we infer that either $v \in X_\alpha \setminus \mathfrak{B}_m(H_\alpha)$ or $w \in Y_\beta \setminus \mathfrak{B}^{\mathrm{rev}}_m(H_\beta)$. We consider the first case, as the second is symmetrical.

Since $w \notin A'$ and $X_\alpha \subseteq A'$, we have that $w \in Y_\alpha$, so vw is an edge in H_α. As $v \notin \mathfrak{B}_m(H_\alpha)$, we have that in H_α vertex v is m-unimportant and has only m-important neighbors. Hence w is m-important in H_α. Observe now that w cannot be contained in $B \setminus A$, as there is more than $m > k$ vertices in X_α being tails of arcs directed toward w, and only k of them can be in separator $A \cap B$ leaving at least one belonging to $A \setminus B$ (recall that vertices from X_α cannot belong to $B \setminus A$). Hence $w \in A$. As $w \notin A'$, we have that $w \in Y_\beta \setminus \mathfrak{B}^{\mathrm{rev}}_m(H_\beta)$. However, w was an m-important vertex on the right side of H_α, so as it is also on the right side of H_β, it is also m-important in H_β. This is a contradiction with $w \notin \mathfrak{B}^{\mathrm{rev}}_m(H_\beta)$.

We conclude that (A', B') is indeed a separation.

Finally, we check that R' is a separation chain. Consider two separations (A_1, B_1), (A_2, B_2) in R, such that $A_1 \subseteq A_2$ and $B_1 \supseteq B_2$. Let $\alpha_1, \beta_1, \alpha_2, \beta_2$ be canonical indices of (A_1, B_1) and (A_2, B_2), respectively. It follows that $\alpha_1 \leq \alpha_2$ and $\beta_1 \leq \beta_2$. Hence, graph H_{α_2} can be obtained from H_{α_1} via a sequence of vertex deletions on the right side and vertex additions on the left side. As $\mathfrak{B}_m$ and $\mathfrak{B}^{\mathrm{rev}}_m$ are monotonic (Lemma 62), by Lemma 55 we have that every vertex deleted from B_1 while constructing B'_1 is also deleted from B_2 while constructing B'_2 (assuming it belongs to B_2). Hence, $B'_2 \subseteq B'_1$. A symmetric argument shows that $A'_2 \supseteq A'_1$. ⌟

We proceed to the algorithm itself. As we have argued, we may look for a trimmed feasible separation chain. Indeed if T admits a path decomposition of width at most k, then by Lemma 32

it admits a feasible separation chain, which by Claim 1 can be turned into a trimmed feasible separation chain. On the other hand, if T admits a trimmed feasible separation chain, then this separation chain can be turned into a path decomposition of T of width at most k using Lemma 32.

Let $\mathcal{N}$ be the family of trimmed separations of T of order at most k. We construct an auxiliary digraph D with vertex set $\mathcal{N}$ by putting an arc $((A,B),(A',B')) \in E(D)$ if and only if $A \subseteq A'$, $B \supseteq B'$ and $|A' \cap B| \leq k+1$. Then paths in D from $(\emptyset, V(T))$ to $(V(T), \emptyset)$ correspond to feasible trimmed separation chains.

We prove that either the algorithm can find an obstacle for admitting path decomposition of width at most k, or D has size at most $2^{\mathcal{O}(k \log k)} \cdot n$ and can be constructed in time $2^{\mathcal{O}(k \log k)} \cdot n^2$. Hence, any linear-time reachability algorithm in D runs within claimed time complexity bound.

Consider any indices α, β such that $\alpha = \max(0, \beta - (4k+1))$. Observe that if $|\mathfrak{B}_m(H_\alpha)| > m^2 + m$, by Lemma 63 we can find a matching of size $m+1$ in H_α. At most $4k+1 = m-k$ edges of this matching have the right endpoint in $Y_\alpha \cap X_\beta$, which leaves us at least $k+1$ edges between X_α and Y_β. Such a structure is a $(k+1,k)$-matching tangle in T, so by Lemma 49 the algorithm may provide a negative answer. A symmetrical reasoning shows that if $|\mathfrak{B}^{\mathrm{rev}}_m(H_\beta)| > m^2 + m$, then the algorithm can also provide a negative answer.

If we assume that these situations do not occur, we can characterize every trimmed separation (A,B) of order at most k by:

- a number β, where $0 \leq \beta \leq n$;
- a mask on the vertices from $Y_\alpha \cap X_\beta$, denoting for each of them whether it belongs to $A \setminus B$, $A \cap B$ or to $B \setminus A$ (at most 3^{4k+1} options);
- subsets of size at most k of $X_\alpha \cap \mathfrak{B}_m(H_\alpha)$ and $Y_\beta \cap \mathfrak{B}^{\mathrm{rev}}_m(H_\beta)$, denoting which vertices belong to $A \cap B$ (at most $\left(k\binom{\mathcal{O}(k^2)}{k}\right)^2 = 2^{\mathcal{O}(k \log k)}$ options).

Moreover, if (A',B') is an outneighbor of (A,B) in D, then it must have parameter β' not larger than $\beta + (5k+2)$, as otherwise we have a guarantee that $|A' \cap B| \geq |X_{\alpha'} \cap Y_\beta| \geq k+2$, and also not smaller than $\beta - (5k+2)$, as otherwise we have a guarantee that $|A| \geq |X_\alpha| > |A'|$. Hence, the outdegrees in D are bounded by $2^{\mathcal{O}(k \log k)}$.

This gives raise to the following algorithm constructing D in time $2^{\mathcal{O}(k \log k)} \cdot n^2$.

- First, we enumerate $\mathcal{N}$. We scan through the order σ with an index β maintaining graphs H_α, H_β for $\alpha = \max(0, \beta - (4k+1))$, along with $\mathfrak{B}_m(H_\alpha)$ and $\mathfrak{B}^{\mathrm{rev}}_m(H_\beta)$. Whenever cardinality of any of these sets exceeds $m^2 + m$, we terminate the algorithm providing a negative answer. By Lemma 64 we can bound the update time by $\mathcal{O}(kn)$. For given index β, we list all $2^{\mathcal{O}(k \log k)}$ pairs (A,B) having this particular β in characterization from the previous paragraph. For every such pair, in $\mathcal{O}(kn)$ we check whether it induces a separation of order k, by testing emptiness of set $E(A \setminus B, B \setminus A)$ using at most $\mathcal{O}(k)$ operations of vertex deletion/addition on the graph H_α. We discard all the pairs that do not form such a separation; all the remaining ones are exactly separations that are trimmed with respect to (α, β).
- For every separation (A,B) characterized by parameter β, we check for all the $2^{\mathcal{O}(k \log k)}$ separations (A',B') with parameter β' between $\beta - (5k+2)$ and $\beta + (5k+2)$, whether we should put an arc from (A,B) to (A',B'). Each such a check can be performed in $\mathcal{O}(k)$ time, assuming that we store the separator along with the separation.

Having constructed D, we run a linear-time reachability algorithm to check whether $(V(T), \emptyset)$ can be reached from $(\emptyset, V(T))$. If not, we provide a negative answer; otherwise, the path corresponds to a feasible separation chain which can be transformed into a path decomposition in $\mathcal{O}(kn)$ time. □

6.3 Algorithms for cutwidth and other ordering problems

In this section we present the algorithms for computing cutwidth and related problems. We start with the approximation algorithm for cutwidth, which follows immediately from the results of Section 6.1. Then we proceed to describing the exact algorithms. It appears that using our approach it is possible to show subexponential parameterized algorithm not only for cutwidth, but also for two related problems, namely Feedback Arc Set and Optimal Linear Arrangement. Both of these problems have been discussed in Section 5.1, so after defining them formally we proceed to explanation of the approach.

6.3.1 Approximation of cutwidth

Results of Section 6.1 immediately yield the following theorem.

Theorem 67. *Let T be a semi-complete digraph and let $m(t) = 64t^2 + 18t + 1$. Then any outdegree ordering of $V(T)$ has width at most $m(\mathbf{ctw}(T))$.*

Proof. Let σ be any outdegree ordering of $V(T)$. If σ had width more than $m(\mathbf{ctw}(T))$, then one of the partitions $(\sigma[\alpha], V(T) \setminus \sigma[\alpha])$ would be a $(m(\mathbf{ctw}(T)) + 1)$-backward tangle. Existence of such a structure is a contradiction with Lemma 52. □

This gives raise to a straightforward approximation algorithm for cutwidth of a semi-complete digraph that simply sorts the vertices with respect to outdegrees, and then scans through the ordering checking whether it has small width. Note that this scan may be performed in $\mathcal{O}(|V(T)|^2)$ time, as we maintain the cutset between the prefix and the suffix of the ordering by iteratively moving one vertex from the suffix to the prefix.

Theorem 68. *There exists an algorithm which, given a semi-complete digraph T on n vertices and an integer k, in time $\mathcal{O}(n^2)$ outputs an ordering of $V(T)$ of width at most $m(k)$ or a $(m(k)+1)$-backward tangle in T, where $m(t) = 64t^2 + 18t + 1$. In the second case the algorithm concludes that $\mathbf{ctw}(T) > k$.*

6.3.2 Additional problem definitions

We now introduce the formal definitions of problems Feedback Arc Set and Optimal Linear Arrangement, and prove their basic properties that will be useful in the algorithms.

Feedback Arc Set

Definition 69. *Let T be a digraph. A subset $F \subseteq E(T)$ is called a* feedback arc set *if $T \setminus F$ is acyclic.*

The Feedback Arc Set problem in semi-complete digraphs is then defined as follows.

Feedback Arc Set (FAS) in semi-complete digraphs	
Input:	A semi-complete digraph T and a nonnegative integer k.
Parameter:	k
Question:	Is there a feedback arc set of T of size at most k?

We have the following easy observation that enables us to view FAS as a graph layout problem.

Lemma 70. *Let T be a digraph. Then T admits a feedback arc set of size at most k if and only if there exists an ordering $(v_1, v_2, \ldots, v_n)$ of $V(T)$ such that at most k arcs of $E(T)$ are directed forward in this ordering, i.e., are of form (v_i, v_j) for $i < j$.*

Proof. If F is a feedback arc set in T then the ordering can be obtained by taking any reverse topological ordering of $T \setminus F$. On the other hand, given the ordering we may simply define F to be the set of forward edges. □

Optimal Linear Arrangement

Definition 71. *Let T be a digraph and $(v_1, v_2, \ldots, v_n)$ be an ordering of its vertices. Then the* cost *of this ordering is defined as*

$$\sum_{(v_i,v_j)\in E(T)} (j-i)\cdot[j>i],$$

that is, every arc directed forwards in the ordering contributes to the cost with the distance between the endpoints in the ordering.

Whenever the ordering is clear from the context, we also refer to the contribution of a given arc to its cost as to the *length* of this arc. By a simple reordering of the computation we obtain the following:

Lemma 72. *For a digraph T and ordering $(v_1, v_2, \ldots, v_n)$ of $V(T)$, the cost of this ordering is equal to:*

$$\sum_{t=1}^{n-1} |E(\{v_1, v_2, \ldots, v_t\}, \{v_{t+1}, v_{t+2}, \ldots, v_n\})|.$$

Proof. Observe that

$$\begin{aligned}
\sum_{(v_i,v_j)\in E(T)} (i-j)\cdot[i>j] &= \sum_{(v_i,v_j)\in E(T)} \sum_{t=1}^{n-1} [i \le t < j] \\
&= \sum_{t=1}^{n-1} \sum_{(v_i,v_j)\in E(T)} [i \le t < j] \\
&= \sum_{t=1}^{n-1} |E(\{v_1, v_2, \ldots, v_t\}, \{v_{t+1}, v_{t+2}, \ldots, v_n\})|.
\end{aligned}$$

□

The problem Optimal Linear Arrangement in semi-complete digraphs is then defined as follows.

Optimal Linear Arrangement (OLA) in semi-complete digraphs	
Input:	A semi-complete digraph T and a nonnegative integer k.
Parameter:	k
Question:	Is there an ordering of $V(T)$ of cost at most k?

6.3.3 k-cuts of semi-complete digraphs

In this section we provide all the relevant observations on k-cuts of semi-complete digraphs. We start with the definitions and enumeration algorithms for k-cuts, and then proceed to bounding the number of k-cuts when the given semi-complete digraph is close to a structured one.

Enumerating k-cuts

Definition 73. *A* k-cut *of a multidigraph T is a partition (X, Y) of $V(T)$ with the following property: there are at most k arcs $(u, v) \in E(T)$ such that $u \in X$ and $v \in Y$. For a multidigraph T, by $\mathcal{N}(T, k)$ we denote the family of all the k-cuts of T.*

The following lemma shows that k-cuts can be enumerated efficiently.

Lemma 74. *Let D be a multidigraph and let X_0, Y_0 be disjoint sets of vertices of D. Then the family of all the k-cuts (X, Y) such that $X_0 \subseteq X$ and $Y_0 \subseteq Y$ can be enumerated with polynomial-time delay, where each k-cut is enumerated together with number $|E(X, Y)|$.*

Proof. Let $\sigma = (v_1, v_2, \ldots, v_p)$ be an arbitrary ordering of vertices of $V(D) \setminus (X_0 \cup Y_0)$. We perform a classical branching strategy. We start with $X = X_0$ and $Y = Y_0$, and consider the vertices in order σ, at each step branching into one of the two possibilities: vertex v_i is to be incorporated into X or into Y. However, after assigning each consecutive vertex we run a max-flow algorithm from X to Y to find the size of a minimum edge cut between X and Y. If this size is more than k, we terminate the branch as we know that it cannot result in any solutions found. Otherwise we proceed. We output a partition after the last vertex, v_n, is assigned a side; note that the last max-flow check ensures that the output partition is actually a k-cut, and finds the output size of the cut as well. Moreover, as during the algorithm we consider only branches that can produce at least one k-cut, the next partition will be always found within polynomial waiting time, proportional to the depth of the branching tree times the time needed for computations at each node of the branching tree. □

Setting $X_0 = Y_0 = \emptyset$ gives an algorithm enumerating k-cuts of D with polynomial-time delay. The running time of Lemma 74 is unfortunately not satisfactory if one would like to design a linear-time algorithm. Therefore, we prove that k-cuts of a semi-complete digraph of small cutwidth can be enumerated more efficiently.

Lemma 75. *There exists an algorithm that, given a semi-complete digraph T on n vertices together with nonnegative integers k and B, works in $\mathcal{O}(n^2 + B \cdot k^{\mathcal{O}(1)} \cdot n)$, and either:*

- *correctly concludes that* $\mathbf{ctw}(T) > k$*;*

- *correctly concludes that the number of k-cuts of T is more than B;*
- *or outputs the whole family $\mathcal{N}(T,k)$ of k-cuts of T with a guarantee that $|\mathcal{N}(T,k)| \leq B$, where each k-cut (X,Y) is output together with $|E(X,Y)|$.*

Proof. If $n \leq 8k+1$, we run the enumeration algorithm of Lemma 74 and terminate it if the number of enumerated k-cuts exceeds B. Since k-cuts are enumerated with polynomial delay and $n = \mathcal{O}(k)$, the algorithm works in $\mathcal{O}(B \cdot k^{\mathcal{O}(1)})$ time.

Assume then that $n > 8k+1$. First, the algorithm computes in $\mathcal{O}(n^2)$ time any outdegree ordering $\sigma = (v_1, v_2, \ldots, v_n)$ of T and all the outdegrees in T. We check in $\mathcal{O}(n)$ time, whether there exists an index i, $1 \leq i \leq n-8k-1$, such that $d^+(v_{i+8k+1}) \leq d^+(v_i)+2k$. If such an index i is found, we infer that $\{v_i, v_{i+1}, \ldots, v_{i+8k+1}\}$ a $(8k+2, 2k)$-degree tangle, implying that $\mathbf{pw}(T) > 2k$ by Lemma 46 and, consequently, that $\mathbf{ctw}(T) > k$ by Lemma 33. Hence, in this case the algorithm can conclude that $\mathbf{ctw}(T) > k$. We proceed with the assumption that this did not take place, i.e., $d^+(v_{i+8k+1}) > d^+(v_i) + 2k$ for all $1 \leq i \leq n - 8k - 1$.

For an index i, $1 \leq i \leq n-8k$, let us define a multidigraph H_i as follows. Start with $T[\{v_i, v_{i+1}, \ldots, v_{i+8k}\}]$ and add two vertices: v_{prefix} and v_{suffix} that correspond to the prefix $\sigma[i-1]$ and suffix $V(T) \setminus \sigma[i+8k]$. For every $j \in \{i, i+1, \ldots, i+8k\}$, add $\min(k+1, |E(\sigma[i-1], \{v_j\})|)$ arcs from v_{prefix} to v_j, and $\min(k+1, |E(\{v_j\}, V(T) \setminus \sigma[i+8k])|)$ arcs from v_j to v_{suffix}. Finally, add $\min(k+1, |E(\sigma[i-1], V(T) \setminus \sigma[i+8k])|)$ arcs from v_{prefix} to v_{suffix}. Note that H_i defined in this manner has size polynomial in k.

The algorithm proceeds as follows. We iterate through consecutive indices i, maintaining the graph H_i. Observe that H_i can be maintained with $\mathcal{O}(n)$ update times, since each update requires inspection of incidence relation between v_i and the whole $V(T)$, and between v_{i+8k+1} and the whole $V(T)$. Thus, the total time spent on maintaining H_i is $\mathcal{O}(n^2)$. For each index i, we enumerate all the k-cuts (X', Y') of H_i such that $v_{\text{prefix}} \in X'$ and $v_{\text{suffix}} \in Y'$ using Lemma 74. This enumeration takes time proportional to their number times a polynomial of the size of H_i, that is, times a polynomial of k. For each enumerated k-cut (X', Y') we construct in $\mathcal{O}(n)$ time one k-cut (X, Y) of T equal to $(\sigma[i-1] \cup (X' \setminus v_{\text{prefix}}), (Y' \setminus v_{\text{suffix}}) \cup (V(T) \setminus \sigma[i+8k])$. By the definition of H_i we infer that (X, Y) is indeed a k-cut of T, and moreover $|E(X,Y)| = |E(X',Y')|$, so the size of the cut output by the algorithm of Lemma 74 can be stored as $|E(X,Y)|$. We store all the k-cuts constructed so far as binary vectors of length n in a prefix tree (trie). Thus in $\mathcal{O}(n)$ time we can check whether (X, Y) has not been already found, in which case it should be ignored, and otherwise we add it to the prefix tree in $\mathcal{O}(n)$ time. If the total number of constructed k-cuts exceed B at any point of the construction, we terminate the algorithm and provide the answer that $|\mathcal{N}(T,k)| > B$. Otherwise, we output all the constructed k-cuts. Since maintenance of graph H_i take $\mathcal{O}(n^2)$ time in total, and each next k-cut is identified and constructed within time $\mathcal{O}(k^{\mathcal{O}(1)} + n)$, for the claimed running time it suffices to show that each k-cut of T is found in this procedure at most $\mathcal{O}(k)$ times.

It remains to argue that (i) in case when the algorithm is providing the family of k-cuts, in fact every k-cut of T is contained in this family, (ii) each k-cut of T is constructed at most $\mathcal{O}(k)$ times. We prove both of the claims at the same time. Let then (X, Y) be a k-cut of T and without loss of generality assume that X and Y are nonempty, since k-cuts $(\emptyset, V(T))$ and $(V(T), \emptyset)$ are enumerated exactly once, for $i = 1$ and $i = n-8k$ respectively. Let α be the maximum index of a vertex of X in σ, and β be the minimum index of a vertex of Y in σ. We claim that $\beta - 1 \leq \alpha \leq \beta + 8k$. Observe that if this claim is proven, then both conditions (i) and (ii) follow: cut (X, Y) is constructed exactly when considering indices i such that $i \leq \beta$ and $i + 8k \geq \alpha$, and the claimed inequalities show that the number of such indices is between 1 and $8k+2$.

We first show that $\beta - 1 \leq \alpha$. Consider two cases. If $\beta = 1$, then in fact all the elements of X have indices larger than β, so in particular $\alpha > \beta$. Otherwise $\beta > 1$, and by the minimality of β we have that $x_{\beta-1} \in X$. Consequently, $\alpha \geq \beta - 1$.

We are left with proving that $\alpha \leq \beta + 8k$. For the sake of contradiction assume that $\alpha > \beta + 8k$, so in particular $1 \leq \beta \leq n - 8k - 1$. By the assumption that $d^+(v_{i+8k+1}) > d^+(v_i) + 2k$ for all $1 \leq i \leq n - 8k - 1$, we have that $d^+(v_\alpha) \geq d^+(v_{\beta+8k+1}) > d^+(v_\beta) + 2k$. By Lemma 20 we infer that there exist $2k$ vertex-disjoint paths of length 2 from v_α to v_β. Since $v_\alpha \in X$ and $v_\beta \in Y$, each of these paths must contain an arc from $E(X, Y)$. This is a contradiction with the assumption that (X, Y) is a k-cut. □

In our dynamic programming algorithms we need to define what does it mean that one k-cut is a possible successor of another k-cut. This is encapsulated in the following definition.

Definition 76. *Let T be a digraph and (X_1, Y_1) and (X_2, Y_2) be two partitions of $V(G)$. We say that cut (X_2, Y_2)* extends cut (X_1, Y_1) using vertex v *if there is one vertex $v \in Y_1$ such that $X_2 = X_1 \cup \{v\}$ and, equivalently, $Y_2 = Y_1 \setminus \{v\}$.*

The following lemma shows that the relation of extension can be computed efficiently within the enumeration algorithm of Lemma 75.

Lemma 77. *If the algorithm of Lemma 75, run on a semi-complete digraph T for parameters k, B, provides the family $\mathcal{N}(T, k)$, then for each k-cut of T there are at most $8k + 1$ k-cuts of T that extend it. Moreover, the algorithm of Lemma 75 can within the same running time in addition construct for each k-cut of T a list of pointers to all the k-cuts that extend it, together with vertices used in these extensions.*

Proof. We only add one additional subroutine to the algorithm of Lemma 75 which computes the lists after the enumeration has been concluded. Assume that during enumeration we have constructed a k-cut (X, Y). Let α be the maximum index of a vertex of X in σ, and let β be the minimum index of a vertex of Y in σ (we take $\alpha = 0$ if $X = \emptyset$ and $\beta = n + 1$ if $Y = \emptyset$). In the proof of Lemma 75 we have proven that $\beta - 1 \leq \alpha \leq \beta + 8k$ (this claim is trivial for $X = \emptyset$ or $Y = \emptyset$), unless the algorithm already provided a negative answer. Let (X', Y') be a k-cut that extends (X, Y), and let v_γ be the vertex used in this extension; thus $\{v_\gamma\} = X' \cap Y$. Define indices α', β' in the same manner for (X', Y'). Note that they satisfy the same inequality, i.e. $\beta' - 1 \leq \alpha' \leq \beta' + 8k$, and moreover $\alpha \leq \alpha'$ and $\beta \leq \beta'$.

We now claim that $\beta \leq \gamma \leq \beta + 8k$. The first inequality follows from the fact that $v_\gamma \in Y$. For the second inequality, assume for the sake of contradiction that $\gamma > \beta + 8k$. Then $\beta \neq \gamma$, and since v_γ is the only vertex that belongs to Y but not to Y', we have that $v_\beta \in Y'$. We infer that $\beta' = \beta$. On the other hand, $v_\gamma \in X'$ implies that $\alpha' \geq \gamma$. Therefore

$$\beta' = \beta < \gamma - 8k \leq \alpha' - 8k,$$

which is a contradiction with the fact that $\alpha' \leq \beta' + 8k$.

Hence, there are only $8k + 1$ possible candidates for k-cuts that extend (X, Y), that is $(X \cup \{v_\gamma\}, Y \setminus \{v_\gamma\})$ for $\beta \leq \gamma \leq \beta + 8k$ and $v_\gamma \in Y$. For each of these candidates we may test in $\mathcal{O}(n)$ time whether it belongs to enumerated family $\mathcal{N}(T, k)$, since $\mathcal{N}(T, k)$ is stored in a prefix tree; note that computing index β also takes $\mathcal{O}(n)$ time. Hence, construction of the lists takes additional $\mathcal{O}(B \cdot k \cdot n)$ time. □

k-cuts of a transitive tournament and partition numbers

For a nonnegative integer n, a partition of n is a multiset of positive integers whose sum is equal to n. The partition number $p(n)$ is equal to the number of different partitions of n. Partition numbers were studied extensively in analytic combinatorics, and there are sharp estimates on their values. In particular, we will use the following:

Lemma 78 ([122, 186])**.** *There exists a constant A such that for every nonnegative k it holds that $p(k) \leq \frac{A}{k+1} \cdot \exp(C\sqrt{k})$, where $C = \pi\sqrt{\frac{2}{3}}$.*

We remark that the original proof of Hardy and Ramanujan [186] shows moreover that the optimal constant A tends to $\frac{1}{4\sqrt{3}}$ as k goes to infinity, i.e., $\lim_{k\to+\infty} \frac{p(k)\cdot(k+1)}{\exp(C\sqrt{k})} = \frac{1}{4\sqrt{3}}$. From now on, we adopt constants A, C given by Lemma 78 in the notation. We use Lemma 78 to obtain the following result, which is the core observation of this section.

Lemma 79. *Let T be a transitive tournament on n vertices and k be a nonnegative integer. Then T has at most $A \cdot \exp(C\sqrt{k}) \cdot (n+1)$ k-cuts, where A, C are defined as in Lemma 78.*

Proof. We prove that for any number a, $0 \leq a \leq n$, the number of k-cuts (X, Y) such that $|X| = a$ and $|Y| = n - a$, is bounded by $A \cdot \exp(C\sqrt{k})$; summing through all the possible values of a proves the claim.

We naturally identify the vertices of T with numbers $1, 2, \ldots, n$, such that arcs of T are directed from larger numbers to smaller, i.e., we order the vertices as in the reversed topological ordering of T. Let us fix some k-cut (X, Y) such that $|X| = a$ and $|Y| = n - a$. Let $x_1 < x_2 < \ldots < x_a$ be the vertices of X.

Let $m_i = x_{i+1} - x_i - 1$ for $i = 0, 1, \ldots, a$; we use convention that $x_0 = 0$ and $x_{a+1} = n + 1$. In other words, m_i is the number of elements of Y that are between two consecutive elements of X. Observe that every element of Y between x_i and x_{i+1} is the head of exactly $a - i$ arcs directed from X to Y: the tails are $x_{i+1}, x_{i+2}, \ldots, x_a$. Hence, the total number of arcs directed from X to Y is equal to $k' = \sum_{i=0}^{a} m_i \cdot (a - i) = \sum_{i=0}^{a} m_{a-i} \cdot i \leq k$.

We define a partition of k' as follows: we take m_{a-1} times number 1, m_{a-2} times number 2, and so on, up to m_0 times number a. Clearly, a k-cut of T defines a partition of k' in this manner. We now claim that knowing a and the partition of k', we can uniquely reconstruct the k-cut (X, Y) of T, or conclude that this is impossible. Indeed, from the partition we obtain all the numbers $m_0, m_1, \ldots, m_{a-1}$, while m_a can be computed as $(n - a) - \sum_{i=0}^{a-1} m_i$. Hence, we know exactly how large must be the intervals between consecutive elements of X, and how far is the first and the last element of X from the respective end of the ordering, which uniquely defines sets X and Y. The only possibilities of failure during reconstruction are that (i) the numbers in the partition are larger than a, or (ii) computed m_a turns out to be negative; in these cases, the partition does not correspond to any k-cut. Hence, we infer that the number of k-cuts of T having $|X| = a$ and $|Y| = n - a$ is bounded by the sum of partition numbers of nonnegative integers smaller or equal to k, which by Lemma 78 is bounded by $(k+1) \cdot \frac{A}{k+1} \cdot \exp(C\sqrt{k}) = A \cdot \exp(C\sqrt{k})$. □

k-cuts of semi-complete digraphs with a small FAS

We have the following simple fact.

Lemma 80. *Assume that T is a semi-complete digraph with a feedback arc set F of size at most k. Let T' be a transitive tournament on the same set of vertices, with vertices ordered as in any topological ordering of $T \setminus F$. Then every k-cut of T is also a $2k$-cut of T'.*

Proof. The claim follows directly from the observation that if (X, Y) is a k-cut in T, then at most k additional arcs directed from X to Y can appear after introducing arcs in T' in place of deleted arcs from F. □

From Lemmata 79 and 80 we obtain the following corollary.

Corollary 81. *If T is a semi-complete digraph with n vertices that has a feedback arc set of size at most k, then the number of k-cuts of T is bounded by $A \cdot \exp(C\sqrt{2k}) \cdot (n+1)$.*

k-cuts of semi-complete digraphs of small cutwidth

To bound the number of k-cuts of semi-complete digraphs of small cutwidth, we need the following auxiliary combinatorial result.

Lemma 82. *Let (X, Y) be a partition of $\{1, 2, \ldots, n\}$ into two sets. We say that a pair (a, b) is* bad *if $a < b$, $a \in Y$ and $b \in X$. Assume that for every integer t there are at most k bad pairs (a, b) such that $a \leq t < b$. Then the total number of bad pairs is at most $k(1 + \ln k)$.*

Proof. Let $y_1 < y_2 < \ldots < y_p$ be the elements of Y. Let m_i be equal to the total number of elements of X that are greater than y_i. Note that m_i is exactly equal to the number of bad pairs whose first element is equal to y_i, hence the total number of bad pairs is equal to $\sum_{i=1}^{p} m_i$. Clearly, sequence (m_i) is non-increasing, so let p' be the last index for which $m_{p'} > 0$. We then have that the total number of bad pairs is equal to $\sum_{i=1}^{p'} m_i$. Moreover, observe that $p' \leq k$, as otherwise there would be more than k bad pairs (a, b) for which $a \leq y_{p'} < b$: for a we can take any y_i for $i \leq p'$ and for b we can take any element of X larger than $y_{p'}$.

We claim that $m_i \leq k/i$ for every $1 \leq i \leq p'$. Indeed, observe that there are exactly $i \cdot m_i$ bad pairs (a, b) for $a \leq y_i$ and $b > y_i$: a can be chosen among i distinct integers $y_1, y_2, \ldots, y_i$, while b can be chosen among m_i elements of X larger than y_i. By the assumption we infer that $i \cdot m_i \leq k$, so $m_i \leq k/i$. Concluding, we have that the total number of bad pairs is bounded by $\sum_{i=1}^{p'} m_i \leq \sum_{i=1}^{p'} k/i = k \cdot H(p') \leq k \cdot H(k) \leq k(1 + \ln k)$, where $H(k) = \sum_{i=1}^{k} 1/i$ is the harmonic function. □

The following claim applies Lemma 82 to the setting of semi-complete digraphs.

Lemma 83. *Assume that T is a semi-complete digraph on n vertices that admits an ordering of vertices $(v_1, v_2, \ldots, v_n)$ of width at most k. Let T' be a transitive tournament on the same set of vertices, where $(v_i, v_j) \in E(T')$ if and only if $i > j$. Then every k-cut of T is a $2k(1+\ln 2k)$-cut of T'.*

Proof. Without loss of generality we assume that T is in fact a tournament, as deleting any of two opposite arcs connecting two vertices can only make the set of k-cuts of T larger, and does not increase the width of the ordering.

Identify vertices $v_1, v_2, \ldots, v_n$ with numbers $1, 2, \ldots, n$. Let (X, Y) be a k-cut of T. Note that arcs of T' directed from X to Y correspond to bad pairs in the sense of Lemma 82: every arc $(b, a) \in E(T')$ such that $a < b$, $a \in Y$, and $b \in X$, corresponds to a bad pair (a, b), and vice versa. Therefore, by Lemma 82 it suffices to prove that for every integer t, the number of arcs $(b, a) \in E(T')$ such that $a \leq t < b$, $a \in Y$, and $b \in X$, is bounded by $2k$. We know that the number

of such arcs in T is at most k, as there are at most k arcs directed from X to Y in T in total. Moreover, as the considered ordering of T has cutwidth at most k, at most k arcs between vertices from $\{1, 2, \ldots, t\}$ and $\{t+1, \ldots, n\}$ can be directed in different directions in T and in T'. We infer that the number of arcs $(b, a) \in E(T')$ such that $a \leq t < b$, $a \in Y$, and $b \in X$, is bounded by $2k$, and so the lemma follows. □

From Lemmata 79 and 83 we obtain the following corollary.

Corollary 84. *Every semi-complete digraph on n vertices and of cutwidth at most k, has at most $A \cdot \exp(2C\sqrt{k(1+\ln 2k)}) \cdot (n+1)$ k-cuts.*

k-cuts of semi-complete digraphs with an ordering of small cost

We firstly show the following lemma that proves that semi-complete digraphs with an ordering of small cost have even smaller cutwidth.

Lemma 85. *Let T be a semi-complete digraph on n vertices that admits an ordering $(v_1, v_2, \ldots, v_n)$ of cost at most k. Then the width of this ordering is at most $(4k)^{2/3}$.*

Proof. We claim that for every integer $t \geq 0$, the number of arcs in T directed from the set $\{v_1, \ldots, v_t\}$ to $\{v_{t+1}, \ldots, v_n\}$ is at most $(4k)^{2/3}$. Let ℓ be the number of such arcs; without loss of generality assume that $\ell > 0$. Observe that at most one of these arcs may have length 1, at most 2 may have length 2, etc., up to at most $\lfloor\sqrt{\ell}\rfloor - 1$ may have length $\lfloor\sqrt{\ell}\rfloor - 1$. It follows that at most $\sum_{i=1}^{\lfloor\sqrt{\ell}\rfloor-1} i \leq \ell/2$ of these arcs may have length smaller than $\lfloor\sqrt{\ell}\rfloor$. Hence, at least $\ell/2$ of the considered arcs have length at least $\lfloor\sqrt{\ell}\rfloor$, so the total sum of lengths of arcs is at least $\frac{\ell \cdot \lfloor\sqrt{\ell}\rfloor}{2} \geq \frac{\ell^{3/2}}{4}$. We infer that $k \geq \frac{\ell^{3/2}}{4}$, which means that $\ell \leq (4k)^{2/3}$. □

Lemma 85 ensures that only $(4k)^{2/3}$-cuts are interesting from the point of view of dynamic programming. Moreover, from Lemma 85 and Corollary 84 we can derive the following statement that bounds the number of states of the dynamic program.

Corollary 86. *If T is a semi-complete digraph with n vertices that admits an ordering of cost at most k, then the number of $(4k)^{2/3}$-cuts of T is bounded by $A \cdot \exp(2C \cdot (4k)^{1/3} \cdot \sqrt{1 + \ln(2 \cdot (4k)^{2/3})}) \cdot (n+1)$.*

6.3.4 The algorithms

We firstly show how using the approach one can find a simple algorithm for Feedback Arc Set.

Theorem 87. *There exists an algorithm that, given a semi-complete digraph T on n vertices and an integer k, in time $\mathcal{O}(\exp(C\sqrt{2k}) \cdot k^{\mathcal{O}(1)} \cdot n^2)$ either finds a feedback arc set of T of size at most k or correctly concludes that this is impossible, where $C = \pi\sqrt{\frac{2}{3}}$.*

Proof. We apply the algorithm of Lemma 75 in T for parameter k and the bound $A \cdot \exp(C\sqrt{2k}) \cdot (n+1)$ given by Corollary 81. If the algorithm concluded that $\mathbf{ctw}(T) > k$, then also the minimum feedback arc set must be of size more than k, and we may provide a negative answer. Similarly, if the algorithm concluded that $|\mathcal{N}(T, k)| > A \cdot \exp(C\sqrt{2k}) \cdot (n+1)$, by Corollary 81 we may also provide a negative answer. Hence, from now on we assume that we have the set $\mathcal{N} := \mathcal{N}(T, k)$

and we know that $|\mathcal{N}| \leq A \cdot \exp(C\sqrt{2k}) \cdot (n+1)$. Note that application of Lemma 75 takes $\mathcal{O}(\exp(C\sqrt{2k}) \cdot k^{\mathcal{O}(1)} \cdot n^2)$ time.

We now describe a dynamic programming procedure that computes the size of optimal feedback arc set basing on the set of k-cuts $\mathcal{N}$. We define an auxiliary weighted digraph D with vertex set $\mathcal{N}$. Intuitively, a vertex from $\mathcal{N}$ corresponds to a partition into prefix and suffix of the ordering.

We define arcs of D as follows. For every pair of k-cuts (X_1, Y_1), (X_2, Y_2) such that (X_2, Y_2) extends (X_1, Y_1) using vertex v, we put an arc from cut (X_1, Y_1) to cut (X_2, Y_2), where the weight of this arc is equal to $|E(X_1, \{v\})|$, that is, the number of arcs that cease to be be directed from the left side to the right side of the partition when moving v between these parts. Construction of arcs D may be performed in $\mathcal{O}(|\mathcal{N}| \cdot k^{\mathcal{O}(1)} \cdot n)$ time, assuming that the enumeration of Lemma 75 constructed also extension lists using Lemma 77. Moreover, the weight of each arc can be computed in $\mathcal{O}(n)$ time by examining outneighbors of v, hence the total time spent on computing weights of arcs is $\mathcal{O}(|\mathcal{N}| \cdot k \cdot n)$. Summarizing, the whole digraph D may be constructed in $\mathcal{O}(|\mathcal{N}| \cdot k^{\mathcal{O}(1)} \cdot n) = \mathcal{O}(\exp(C\sqrt{2k}) \cdot k^{\mathcal{O}(1)} \cdot n^2)$ time, and has $|\mathcal{N}|$ vertices and $\mathcal{O}(|\mathcal{N}| \cdot k)$ arcs.

Observe that a path from vertex $(\emptyset, V(T))$ to a vertex $(V(T), \emptyset)$ of total weight ℓ defines an ordering of vertices of T that has exactly ℓ forward arcs — each of these arcs was taken into account while moving its head from the right side of the partition to the left side. On the other hand, every ordering of vertices of T that has exactly $\ell \leq k$ forward arcs defines a path from $(\emptyset, V(T))$ to $(V(T), \emptyset)$ in D of total weight ℓ; note that all partitions into prefix and suffix in this ordering are k-cuts, so they constitute legal vertices in D. Hence, we need to check whether vertex $(V(T), \emptyset)$ can be reached from $(\emptyset, V(T))$ by a path of total weight at most k. This, however, can be done in time $\mathcal{O}((|V(D)| + |E(D)|) \log |V(D)|) = \mathcal{O}(\exp(C\sqrt{2k}) \cdot k^{\mathcal{O}(1)} \cdot n \log n)$ using Dijkstra's algorithm. The feedback arc set of size at most k can be easily retrieved from the constructed path in $\mathcal{O}(n^2)$ time. □

We remark that it is straightforward to adapt the algorithm of Theorem 87 to the weighted case, where all the arcs are assigned a real weight larger or equal to 1 and we parametrize by the target total weight of the solution. As the minimum weight is at least 1, we may still consider only k-cuts of the digraph where the weights are forgotten. On this set we employ a modified dynamic programming routine, where the weights of arcs in digraph D are not simply the number of arcs in $E(\{v\}, X_1)$, but their total weight.

We now proceed to the main result of this section, i.e., the subexponential algorithm for cutwidth of a semi-complete digraph.

Theorem 88. *There exists an algorithm that, given a semi-complete digraph T on n vertices and an integer k, in time $2^{\mathcal{O}(\sqrt{k \log k})} \cdot n^2$ either computes a vertex ordering of width at most k or correctly concludes that this is impossible.*

Proof. Apply the algorithm of Lemma 75 in T for parameter k and the bound $A \cdot \exp(2C\sqrt{k(1+\ln 2k)}) \cdot (n+1) = 2^{\mathcal{O}(\sqrt{k \log k})} \cdot n$ given by Corollary 84. If the algorithm concluded that $\mathbf{ctw}(T) > k$, then we may provide a negative answer. Similarly, if the algorithm concluded that $|\mathcal{N}(T,k)| > A \cdot \exp(2C\sqrt{k(1+\ln 2k)}) \cdot (n+1)$, by Corollary 84 we may also providing a negative answer. Hence, from now on we assume that we have the set $\mathcal{N} := \mathcal{N}(T,k)$ and we know that $|\mathcal{N}| \leq 2^{\mathcal{O}(\sqrt{k \log k})} \cdot n$. Note that application of Lemma 75 takes $\mathcal{O}(2^{\mathcal{O}(\sqrt{k \log k})} \cdot n^2)$ time.

We now describe a dynamic programming routine that basing on the set $\mathcal{N}$ computes an ordering of width at most k, or correctly concludes that it is impossible. The routine is very similar to that of Theorem 87, so we describe only the necessary modifications.

We define an auxiliary digraph D on the vertex set $\mathcal{N}$ exactly in the same manner as in the proof of Theorem 87, but this time D is unweighted. Similarly as before, D may be constructed in time $\mathcal{O}(|\mathcal{N}| \cdot k^{\mathcal{O}(1)} \cdot n) = 2^{\mathcal{O}(\sqrt{k \log k})} \cdot n^2$, and has $|\mathcal{N}|$ vertices and $\mathcal{O}(|\mathcal{N}| \cdot k)$ arcs. Clearly, paths in D from $(\emptyset, V(T))$ to $(V(T), \emptyset)$ correspond to orderings of $V(T)$ of cutwidth at most k. Therefore, it suffices check whether in D there exists a path from $(\emptyset, V(T))$ to $(V(T), \emptyset)$ using depth-first search, which takes $\mathcal{O}(|V(D)| + |E(D)|) = 2^{\mathcal{O}(\sqrt{k \log k})} n$ time. □

Finally, we present how the framework can be applied to the OLA problem.

Theorem 89. *There exists an algorithm that, given a semi-complete digraph T on n vertices and an integer k, in time $2^{\mathcal{O}(k^{1/3}\sqrt{\log k})} \cdot n^2$ either computes a vertex ordering of cost at most k, or correctly concludes that it is not possible.*

Proof. We apply the algorithm of Lemma 75 in T for parameter $(4k)^{2/3}$ and the bound $A \cdot \exp(2C \cdot (4k)^{1/3} \cdot \sqrt{1 + \ln(2 \cdot (4k)^{2/3})}) \cdot (n+1) = 2^{\mathcal{O}(k^{1/3}\sqrt{\log k})} \cdot n$ given by Corollary 86. If the algorithm concluded that $\mathbf{ctw}(T) > (4k)^{2/3}$, then by Lemma 85 we may provide a negative answer. Similarly, if the algorithm concluded that $|\mathcal{N}(T, (4k)^{2/3})| > A \cdot \exp(2C \cdot (4k)^{1/3} \cdot \sqrt{1 + \ln(2 \cdot (4k)^{2/3})}) \cdot (n+1)$, by Corollary 86 we may also provide a negative answer. Hence, from now on we assume that we have the set $\mathcal{N} := \mathcal{N}(T, (4k)^{2/3})$ and we know that $|\mathcal{N}| \leq 2^{\mathcal{O}(k^{1/3}\sqrt{\log k})} \cdot n$. Note that application of Lemma 75 takes $2^{\mathcal{O}(k^{1/3}\sqrt{\log k})} \cdot n^2$ time.

In order to construct the dynamic programming routine, we proceed very similarly to the proof of Theorem 87. Define the same auxiliary digraph D on the vertex set $\mathcal{N}$, where we put an arc from (X_1, Y_1) to (X_2, Y_2) if and only if (X_2, Y_2) extends (X_1, Y_1); this time, the weight of this arc is equal to $|E(X_1, Y_1)|$. As in the proof of Theorem 87, the digraph D may be constructed in time $\mathcal{O}(|\mathcal{N}| \cdot k^{\mathcal{O}(1)} \cdot n)$, and has $|\mathcal{N}|$ vertices and $\mathcal{O}(|\mathcal{N}| \cdot k)$ arcs; note here that we do not need to additionally compute the weights of arcs in D, since values $|E(X, Y)|$ have been provided together with the cuts (X, Y) by the enumeration algorithm.

Now observe that paths from $(\emptyset, V(T))$ to $(V(T), \emptyset)$ of total weight $\ell \leq k$ correspond one-to-one to orderings with cost ℓ: the weight accumulated along the path computes correctly the cost of the ordering due to Lemma 72. Note that Lemma 85 ensures that in an ordering of cost at most k, the only feasible partitions into prefix and suffix of the ordering are in $\mathcal{N}$, so they constitute legal vertices in D. Hence, we may apply Dijkstra's algorithm to check whether vertex $(V(T), \emptyset)$ is reachable from $(\emptyset, V(T))$ via a path of total weight at most k, and this application takes $\mathcal{O}((|V(D)| + |E(D)|) \log |V(D)|) = 2^{\mathcal{O}(k^{1/3}\sqrt{\log k})} \cdot n \log n$ time. The corresponding ordering may be retrieved from this path in $\mathcal{O}(n)$ time. □

Similarly to Theorem 87, it is also straightforward to adapt the algorithm of Theorem 89 to the natural weighted variant of the problem, where each arc is assigned a real weight larger or equal to 1, each arc directed forward in the ordering contributes to the cost with its weight multiplied by the length of the arc, and we parametrize by the total target cost. One needs also to maintain the total weight of the cut in the enumeration algorithm of Lemma 75 to avoid its recomputation for every arc of D, which done by brute-force would increase the running time from quadratic in terms of n to cubic.

6.4 Conclusions and open problems

In this chapter we have presented new methods of computing the two main width measures of semi-complete digraphs: cutwidth and pathwidth. For both of the width measures we have designed a polynomial-time approximation algorithm as well as an FPT exact algorithm. Our algorithms have better guarantees on performance than previous methods mainly thanks to a new set of combinatorial obstacles. We believe that the current work settles good foundations for future research on topological problems in semi-complete digraphs, since computing exactly or approximately a width measure of a combinatorial object often serves as a crucial subroutine in algorithms for other problems.

Nevertheless, the results of this chapter leave a number of thrilling open questions about the complexity of computing the cutwidth and pathwidth of a semi-complete digraph. To begin with, we are still lacking proofs that computing cutwidth and pathwidth of a semi-complete digraph exactly is NP-hard. It is very hard to imagine that any of these problems could be solved in polynomial-time; however, proving NP-hardness of computing width measures is usually technically very challenging. For example, NP-hardness of computing cliquewidth of an undirected graph has been shown only in 2006 by Fellows et al. [129], after resisting attacks for a few years as a long-standing open problem. Furthermore, proving NP-hardness results for problems in tournaments is also known to be extremely difficult, because the instance obtained in the reduction is already very much constrained by the fact that it must be a tournament. The Feedback Arc Set problem, which is a simpler relative of the problem of computing exactly the cutwidth of a tournament, was proven to be NP-hard also only recently. First, Ailon et al. proved that the problem is NP-hard under randomized reductions [5], and then Alon [9] and Charbit et al. [60] independently presented proofs that used only deterministic reductions, thus settling NP-hardness of the problem in the classical sense. Therefore, the author conjectures that it should be very challenging to show that computing exactly the cutwidth of a semi-complete digraph is hard: one needs both to overcome complications that arise when considering a width measure that is defined via existence of a global decomposition, and to solve problems that made the reduction for Feedback Arc Set so elusive. The author's feeling is that settling NP-hardness of computing exactly the pathwidth of a semi-complete digraph should be more attainable, and therefore it is a natural next step.

When it comes to approximation algorithms, the obvious open question is to determine whether cutwidth admits a constant factor approximation; recall that the algorithm presented in this chapter is only an $\mathcal{O}(OPT)$-approximation. As far as pathwidth is concerned, it is natural to ask if the parameter admits a PTAS, or whether computing it is APX-hard. Obviously, we need first to answer the question about NP-hardness, which is still unclear. The author's feeling is that APX-hardness should follow from the NP-hardness reduction.

Finally, the exact algorithms also give much room for improvement. For pathwidth it is natural to ask for an algorithm working in $\mathcal{O}^*(2^{\mathcal{O}(k)})$ time. Currently, the additional $\log k$ factor in the exponent is an artifact of using quadratic kernel of Buss for identifying sets of candidates for consecutive bags. If one was able for every bag to find a set of candidates for elements of this bag which was of size linear in k instead of quadratic, then an algorithm working in time $\mathcal{O}^*(2^{\mathcal{O}(k)})$ would immediately follow. As far as cutwidth is concerned, the $\log k$ factor under the square root in the exponent seems also artificial. At this moment, appearance of this factor is a result of pipelining Lemma 79 with Lemma 82 in the proof of Lemma 83. A closer examination of the proofs of Lemmas 79 and 82 shows that bounds given by them are essentially optimal on their own; yet, it is not clear whether the bound given by pipelining them is optimal as well. The author conjectures that it should be possible to obtain an $\mathcal{O}^*(2^{\mathcal{O}(k)})$ algorithm for pathwidth and an $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k})})$ algorithm

for cutwidth, and that these upper bounds can be matched by tight lower bounds (i.e., no $\mathcal{O}^*(2^{o(k)})$ and $\mathcal{O}^*(2^{o(\sqrt{k})})$ algorithms, respectively) under Exponential-Time Hypothesis. Of course, obtaining such lower bounds would be even more difficult than settling NP-hardness, which is still open.

Chapter 7

Solving topological problems

7.1 Algorithms for containment testing

In this section we utilize the results of the last chapter to give fast algorithms for containment testing. As described in Chapter 5, the WIN/WIN approach employed for containment testing makes use of dynamic programming routines that work on decompositions of bounded width. More precisely, we would like to design a dynamic programming routine that on decomposition of width p of a semi-complete digraph T on n vertices, works in time $f(|H|,p)\cdot n^2$ for some (possibly small) function f. Existence of such an algorithm for the topological subgraph and minor relations follows immediately from Theorem 36, as the definition of containing H as a topological subgraph or a minor can be trivially expressed by a $\mathbf{MSO}_1$ formula of length depending on $|H|$ only. For the immersion relation one can use the following result of Ganian et al. [160].

Theorem 90 ([160])**.** *For every ℓ there exists an $\mathbf{MSO}_1$ formula $\pi_\ell(s_1,s_2,\ldots,s_\ell,t_1,t_2,\ldots,t_\ell)$ that for a digraph with distinguished vertices $s_1,s_2,\ldots,s_\ell,t_1,t_2,\ldots,t_\ell$ (some of which are possibly equal) asserts whether there exists a family of arc-disjoint paths $P_1,P_2,\ldots,P_\ell$ such that P_i begins in s_i and ends in t_i, for $i=1,2,\ldots,\ell$.*

Therefore, in order to construct a formula expressing that a digraph T contains a digraph H as an immersion, we can quantify existentially the images of vertices of H, verify that they are pairwise different, and then use formula $\pi_{|E(H)|}$ to check existence of arc-disjoint paths between corresponding images. Unfortunately, for Theorem 36 only very crude upper bounds on the obtained function f can be given. Essentially, as described in Sections 3.2.2 and 3.3.1, the engine behind Theorem 36 is translation of formulas of **MSO** logic on trees to tree automata. It is known that the dependence of the size of the automaton obtained in the translation on the size of the formula is roughly q-times exponential, where q is the quantifier rank of the formula. Moreover, this is unavoidable in the following sense: the size of the automaton cannot be bounded by any elementary function of the length of the formula. Since formulas expressing the topological subgraph and minor relations have constant quantification rank, the function $f(\cdot,\cdot)$ given by an application of Theorem 36 will be in fact elementary. As far as the immersion relation is concerned, the quantification rank of the formula given by Theorem 90 is linear in ℓ, and thus we cannot even claim elementary running time bounds.

Instead, in order to give explicit bounds on the running time of topological subgraph, immersion, and minor testing, we design explicit dynamic programming routines. The routines are presented in Section 7.4, and here we only state the results. We remark that the routines can in fact check more involved properties, which will be useful in applications.

Theorem 91. *There exists an algorithm that given a digraph H with k vertices, ℓ arcs, a set $F \subseteq E(H)$ of constraints, and a semi-complete digraph T on n vertices together with its path decomposition of width p, checks whether H is topologically contained in T with constraints F in time $2^{\mathcal{O}((p+k+\ell)\log p)} \cdot n$.*

Theorem 92. *There exists an algorithm that, given a rooted digraph $(H; v_1, v_2, \ldots, v_t)$ with k vertices and ℓ arcs, and a semi-complete rooted digraph $(T; w_1, w_2, \ldots, w_t)$ on n vertices together with its path decomposition of width p, checks whether H can be immersed into T while preserving roots in time $2^{\mathcal{O}(k \log p + p\ell(\log \ell + \log p))} \cdot n$.*

It may be somewhat surprising that the algorithms of Theorems 91 and 92 work in time linear in n, while representation of T using adjacency matrix uses $\mathcal{O}(n^2)$ space. Note however, that a representation of a path decomposition of width p uses only $\mathcal{O}(pn)$ space. In these theorems we assume that the algorithm is given access to an adjacency matrix representing T that can be queried in constant time. Both algorithms perform one linear scan through the given path decomposition using the adjacency matrix as a black-box, and performing only a linear number of queries on it.

By pipelining Lemma 22 and Theorem 91 we obtain also the routine for the minor relation.

Theorem 93. *There exists an algorithm that given digraph H with k vertices and ℓ arcs, and a semi-complete digraph T together with its path decomposition of width p, checks whether H is a minor of T in time $2^{\mathcal{O}((p+k+\ell)(\log p + \log k + \log \ell))} \cdot n$.*

We are ready to provide formal descriptions of the containment testing algorithms.

Theorem 94. *There exists an algorithm that, given a semi-complete T on n vertices and a digraph H with $k = |H|$, in time $2^{\mathcal{O}(k \log k)} \cdot n^2$ checks whether H is topologically contained in T.*

Proof. We run the algorithm given by Theorem 65 for parameters $20k$ and $520k$, which either returns a $(520k + 2, 20k)$-degree tangle, a $(20k + 1, 20k)$-matching tangle or a decomposition of width at most $560k$. If the last is true, we run the dynamic programming routine of Theorem 91, which works in $2^{\mathcal{O}(k \log k)} \cdot n$ time. However, if the approximation algorithm returned an obstacle, by Lemmas 47, 50 and 41 we can provide a positive answer: existence of a $(520k + 2, 20k)$-degree tangle or a $(20k + 1, 20k)$-matching tangle ensures that H is topologically contained in T. □

By plugging in the dynamic programming routine for immersion (Theorem 114 with no roots specified) instead of topological containment, we obtain the following:

Theorem 95. *There exists an algorithm that, given a semi-complete T on n vertices and a digraph H with $k = |H|$, in time $2^{\mathcal{O}(k^2 \log k)} \cdot n^2$ checks whether H can be immersed into T.*

Finally, in the same manner we can use the algorithm of Theorem 93 and substitute the usage of Lemma 41 with Lemma 42 to obtain the algorithm for testing the minor relation. Note that application of Lemma 42 requires providing a slightly larger jungle; hence, in the application of Theorem 65 we use parameters $45k$ and $1170k$ instead of $20k$ and $520k$.

Theorem 96. *There exists an algorithm that, given a semi-complete T on n vertices and a digraph H with $k = |H|$, in time $2^{\mathcal{O}(k \log k)} \cdot n^2$ checks whether H is a minor of T.*

7.2 Containment testing and meta-theorems

We observe that FPT algorithms for testing containment relations can be used also to prove meta-theorems of more general nature using the WQO results of Chudnovsky and Seymour [74] and of Kim and Seymour [216]. We explain this on the following example. Let Π be a class of digraphs and denote by $\Pi + kv$ the class of digraphs, from which one can delete at most k vertices to obtain a member of Π. We study the following problem:

$\Pi + kv$ Recognition	
Input:	Digraph D and a non-negative integer k
Parameter:	k
Question:	Is there $S \subseteq V(D)$, $\lvert S\rvert \leq k$, such that $D \setminus S \in \Pi$?

We are interested in classes Π which are closed under immersion. For example the class of acyclic digraphs, or digraphs having cutwidth at most c, where c is some constant, are of this type (see Lemma 26). In particular, the parameterized Feedback Vertex Set in directed graphs is equivalent to $\Pi + kv$ Recognition for Π being the class of acyclic digraphs. Chudnovsky and Seymour [74] showed that immersion order on semi-complete digraphs is a well-quasi-order, see Theorem 23. Based on this result and the approximation algorithm for pathwidth, we are able to prove the following meta-theorem. Note that it seems difficult to obtain the results of this flavor using cutwidth, as cutwidth can decrease dramatically even when one vertex is deleted from the digraph.

Theorem 97. *Let Π be an immersion-closed class of semi-complete digraphs. Then $\Pi+kv$* Recognition *is FPT on semi-complete digraphs.*

Proof. As Π is immersion-closed, by Theorem 23 we infer that Π can be characterized by admitting no member of a family of semi-complete digraphs $\{H_1, H_2, \ldots, H_r\}$ as an immersion, where $r = r(\Pi)$ depends only on the class Π. We will again make use of Theorem 90. For every $i \in \{1, 2, \ldots, r\}$ we construct an $\mathbf{MSO}_1$ formula $\varphi_i(X)$ with one free monadic vertex variable X that is true if and only if digraph $G \setminus X$ contains H_i as an immersion. We simply quantify existentially over the images of vertices of H_i, use the appropriate formula $\pi_{|E(H)|}$ for quantified variables to express existence of arc-disjoint paths, and at the end relativize the whole formula to the subdigraph induced by $V(T) \setminus X$. Hence, if we denote by $\psi_k(X)$ the assertion that $|X| \leq k$ (easily expressible in **FO** by a formula, whose length depends on k), the formula $\varphi = \exists_X \psi_k(X) \wedge \bigwedge_{i=1}^{r} \neg\varphi_i(X)$ is true exactly in semi-complete digraphs from which one can delete at most k vertices in order to obtain a semi-complete digraphs belonging to Π.

Observe that every member of class Π has pathwidth bounded by a constant depending on Π only, as by Theorem 65 and Lemmas 47, 50 and 41, a semi-complete digraph of large enough pathwidth contains a sufficiently large short jungle, in which one of the digraphs H_i is topologically contained, so also immersed. It follows that if the pathwidth of every member of Π is bounded by c_Π, then the pathwidth of every member of $\Pi + kv$ is bounded by $c_\Pi + k$. Therefore, we can apply the following WIN/WIN approach. We apply Theorem 65 for parameters k' and $4k'$ where $k' = c_\Pi + k$. This application takes time $g(k)|V(T)|^2$ and provides either an obstacle for admitting a path decomposition of width at most $c_\Pi + k$, which is sufficient to provide a negative answer, or a path decomposition of width at most $6(c_\Pi + k)$, on which we can run the algorithm given by Theorem 36 applied to formula φ. □

7.3 The algorithm for Rooted Immersion

In this section we apply the developed tools to solve the Rooted Immersion problem in semi-complete digraphs, i.e., testing whether one digraph H is an immersion of another semi-complete digraph T where some of the vertices have already prescribed images.

The algorithm of Theorem 95 cannot be used to solve Rooted Immersion because in the case when an obstacle is found, we are unable to immediately provide the answer: even though H can be found in the obstacle as an immersion, this embedding can have nothing to do with the roots. Therefore, we need to exploit the identified obstacle in a different way. Following the classical approach in such a case, we design an *irrelevant vertex rule*. That is, given the obstacle we find in polynomial time a vertex that can be assumed to be not used by some solution, and which therefore can be safely deleted from the graph. After this deletion we simply restart the algorithm. Thus at each step of this process the algorithm either solves the problem completely using dynamic programming on a path decomposition of small width, or removes one vertex of the graph; as a result, we run the approximation algorithm of Theorem 65 and identification of an irrelevant vertex at most n times.

The design of the irrelevant vertex rule needs very careful argumentation, because one has to argue that some vertex is omitted by some solution without having any idea about the shape of this solution, or even of existence of any solutions at all. Therefore, instead of short jungles that were the main tool used in Section 7.1, we use the original obstacle introduced by Fradkin and Seymour [153], namely the triple.

7.3.1 Irrelevant vertex in a triple

In this section we show how to identify a vertex that is irrelevant for the Rooted Immersion problem in a semi-complete graph with a sufficiently large triple. Let $p(k) = 80k^2 + 80k + 5$. We prove the following theorem.

Theorem 98. *Let $\mathcal{I} = ((H; u_1, u_2, \ldots, u_t), (T; v_1, v_2, \ldots, v_t))$ be an instance of the* Rooted Immersion *problem, where $k = |H|$ and T is a semi-complete graph on n vertices containing a $p(k)$-triple (A, B, C) disjoint with $\{v_1, v_2, \ldots, v_t\}$. Then it is possible in time $\mathcal{O}(p(k)^2 \cdot n^2)$ to identify a vertex $x \in B$ such that $\mathcal{I}$ is a YES instance of* Rooted Immersion *if and only if $\mathcal{I}' = ((H, u_1, u_2, \ldots, u_t), (T \setminus \{x\}, v_1, v_2, \ldots, v_t))$ is a YES instance.*

Before we proceed with the proof of Theorem 98, we need to make several auxiliary observations.

Let η be a solution to the Rooted Immersion instance and let $\mathcal{Q}$ be the family of paths being images of all the arcs in H, i.e., $\mathcal{Q} = \eta(E(H))$. We call an arc (a vertex) *used by a path P* if it is traversed by P. We say that an arc (a vertex) is *used by $\mathcal{Q}$* if it is used by any path of $\mathcal{Q}$. We omit the family $\mathcal{Q}$ whenever it is clear from the context. An arc (a vertex) which is not used is called a *free* arc (vertex).

Observation 99. *If $\mathcal{Q}$ is a family of paths containing simple paths only, then every vertex in T is adjacent to at most k used incoming arcs and at most k used outgoing arcs.*

Proof. Otherwise, there is a path in the solution that visits that vertex at least two times. Therefore, there is a cycle on this path, which contradicts its simplicity. □

Let η be a solution to Rooted Immersion instance $\mathcal{I}$ that minimizes the total sum of length of paths in $\eta(E(H))$, and let $\mathcal{Q} = \eta(E(H))$. Firstly, we observe some easy properties of $\mathcal{Q}$.

Observation 100. *Every path from $\mathcal{Q}$ uses at most 2 arcs from the matching between C and A.*

Proof. Assume otherwise, that there is a path $P \in \mathcal{Q}$ that uses three arcs of the matching: (c_1, a_1), (c_2, a_2), (c_3, a_3), appearing in this order on the path. By Observation 99, for at most k vertices of $v \in B$ the arc (a_1, v) is used. For the same reason, for at most k vertices $v \in B$ the arc (v, c_3) is used. As $|B| > 2k$, there exists $v \in B$ such that (a_1, v) and (v, c_3) are not used. Now replace the part of P appearing between a_1 and c_3 with $a_1 \to v \to c_3$. We obtain a solution with smaller sum of lengths of the paths, a contradiction. □

Observation 101. *Every path from $\mathcal{Q}$ uses at most $2k + 4$ vertices from A.*

Proof. Assume otherwise, that there is a path $P \in \mathcal{Q}$ passing through at least $2k + 5$ vertices from A. By Observation 100, at most $2k$ of them are endpoints of used arcs of the matching between C and A. Therefore, there are at least 5 visited vertices, which are endpoints of an unused arc of the matching. Let us denote any 5 of them by a_1, a_2, a_3, a_4, a_5 and assume that they appear on P in this order. Let (c_5, a_5) be the arc of the matching between C and A that is incident to a_5. By the same reasoning as in the proof of Observation 100, there exists a vertex $v \in B$, such that (a_1, v) and (v, c_5) are unused arcs. Substitute the part of the path P between a_1 and a_5 by the path $a_1 \to v \to c_5 \to a_5$, which consists only of unused arcs. We obtain a solution with smaller sum of lengths of the paths, a contradiction. □

A symmetrical reasoning yields the following observation.

Observation 102. *Every path from $\mathcal{Q}$ uses at most $2k + 4$ vertices from C.*

Finally, we prove a similar property for B.

Observation 103. *Every path from $\mathcal{Q}$ uses at most 4 vertices from B.*

Proof. Assume otherwise, that there is a path $P \in \mathcal{Q}$ such that it passes through at least 5 vertices from B. Let us denote any 5 of them by b_1, b_2, b_3, b_4, b_5 and assume that they appear on P in this order. By Observation 99 there are at most k outgoing arcs incident to b_1 used, and there are at most k incoming arcs incident to b_5 used. Moreover, by Observation 100 there are at most $2k$ arcs of the matching between C and A used. As $p(k) > 4k$, we conclude that there is an unused arc of the matching (c, a), such that arcs (b_1, c) and (a, b_5) are also unused. Substitute the part of the path P between b_1 and b_5 by the path $b_1 \to c \to a \to b_5$. We obtain a solution with smaller sum of lengths of the paths, a contradiction. □

From Observations 101-103 we obtain the following corollary.

Corollary 104. *In B there are at most $4k$ vertices used by $\mathcal{Q}$, so in particular there are at least $5k + 1$ vertices free from $\mathcal{Q}$. Moreover, within the matching between C and A there are at least $4k$ arcs having both endpoints free from $\mathcal{Q}$.*

We note that the Corollary 104 holds also for much larger values than $5k + 1$, $4k$, respectively; we choose to state it in this way to show how many free vertices from B and free arcs of the matching we actually use in the proof of Theorem 98. We need one more auxiliary lemma that will prove to be useful.

Lemma 105. *Let $T = (V_1 \cup V_2, E)$ be a semi-complete bipartite digraph, i.e., a directed graph, where arcs are only between V_1 and V_2, but for every $v_1 \in V_1$ and $v_2 \in V_2$ at least one of the arcs (v_1, v_2) and (v_2, v_1) is present. Then at least one of the assertions holds:*

(a) for every $v_1 \in V_1$ there exists $v_2 \in V_2$ such that $(v_1, v_2) \in E$;

(b) for every $v_2 \in V_2$ there exists $v_1 \in V_1$ such that $(v_2, v_1) \in E$.

Proof. Assume that (a) does not hold. This means that there is some $v_0 \in V_1$ such that for all $v_2 \in V_2$ we have $(v_2, v_0) \in E$. Then we can always pick v_0 as v_1 in the statement of (b), so (b) holds. □

Observe that by reversing all the arcs we can obtain a symmetrical lemma, where we assert existence of inneighbors instead of outneighbors.

We are now ready to prove Theorem 98. Whenever we will refer to the *matching*, we mean the matching between C and A.

Proof of Theorem 98. To prove the theorem we give an algorithm that outputs a vertex $x \in B$, such that if there exists a solution to the the given instance, then there exists also a solution in which no path passes through x. The algorithm will run in time $\mathcal{O}(p(k)^2 \cdot n^2)$.

We proceed in three steps. The first step is to identify in $\mathcal{O}(p(k)^2 \cdot n^2)$ time a set $X \subseteq B$, $|X| \geq 16k^2 + 16k + 1$, such that if $\mathcal{I}$ is a YES instance, then for every $x \in X$ there is a solution η with $\mathcal{P} = \eta(E(H))$ having the following properties:

(1.i) at least $3k + 1$ vertices of B are free from $\mathcal{P}$;

(1.ii) at least $2k$ arcs of the matching have both endpoints free from $\mathcal{P}$;

(1.iii) if x is accessed by some path $P \in \mathcal{P}$ from a vertex v, then $v \in A$.

The second step of the proof is to show that one can identify in $\mathcal{O}(p(k)^2 \cdot n^2)$ time a vertex $x \in X$ such that if $\mathcal{I}$ is a yes instance, then there is a solution with $\mathcal{P} = \eta(E(H))$ having the following properties:

(2.i) at least $k + 1$ vertices of B are free from $\mathcal{P}$;

(2.ii) if x is accessed by some path $P \in \mathcal{P}$ from a vertex v, then $v \in A$;

(2.iii) if x is left by some path $P \in \mathcal{P}$ to a vertex v, then $v \in C$.

The final, concluding step of the proof is to show that there is a solution $\mathcal{P} = \eta(E(H))$ such that

(3.i) No path from $\mathcal{P}$ is using x.

We proceed with the first step. Let $\mathcal{Q} = \eta(E(H))$, where η is the solution for the Rooted Immersion instance with the minimum sum of lengths of the paths.

For every vertex $b \in B$, we identify two sets: G_b, R_b. The set R_b consists of those inneighbors of b outside A, which are inneighbors of at least $6k$ vertices from B, while G_b consists of the rest of inneighbors of b outside A. Formally,

$$R_b = \{v \mid v \in V(T) \setminus A \wedge (v, b) \in E \wedge |N^+(v) \cap B| \geq 6k\},$$
$$G_b = \{v \mid v \in V(T) \setminus A \wedge (v, b) \in E \wedge |N^+(v) \cap B| < 6k\}.$$

Note that R_b, G_b can be computed in $\mathcal{O}(n^2)$ time. Let $B_\emptyset$ be the set of those vertices $b \in B$, for which $G_b = \emptyset$. We claim that if $|B_\emptyset| \geq 16k^2 + 16k + 1$, then we can set $X = B_\emptyset$.

Take any $b \in B_\emptyset$. We argue that we can reroute the paths of $\mathcal{Q}$ that access b from outside A in such a manner, that during rerouting each of them we use at most one additional free vertex from B and at most one additional arc from the matching. We reroute the paths one by one. Take path P that accesses b from outside A, and let v be the previous vertex on the path. As $G_b = \emptyset$, $v \in R_b$. Therefore, v has at least $6k$ outneighbors in B. Out of them, at most $4k$ are not free with respect to $\mathcal{Q}$, due to Observation 103, while at most $k-1$ were used by previous reroutings. Therefore, there is a vertex $b' \in B \cap N^+(v)$, such that b' is still free. Thus we can substitute usage of the arc (v, b) on P by the path $v \to b' \to c \to a \to b$, where (c, a) is an arbitrary arc of the matching that still has both endpoints free, which exists due to using at most $k-1$ of them so far. After the rerouting we examine the obtained walk and remove all the cycles on this walk so that we obtain a simple path. Note that this shortcutting step cannot spoil property (1.iii) for this path.

We are now left with the case when $|B_\emptyset| < 16k^2 + 16k + 1$. Let $B_g = B \setminus B_\emptyset$. Then $|B_g| \geq 4(16k^2 + 16k + 1)$. We construct a semi-complete digraph $S = (B_g, L)$ as follows. For every $b_1, b_2 \in B_g$, $b_1 \neq b_2$, we put arc (b_1, b_2) if for every $v \in G_{b_1}$, either $v \in G_{b_1} \cap G_{b_2}$ or v has an outneighbor in G_{b_2}. Similarly, we put arc (b_2, b_1) into L if for every $v \in G_{b_2}$, either $v \in G_{b_1} \cap G_{b_2}$ or v has an outneighbor in G_{b_1}. By applying Lemma 105 to the semi-complete bipartite graph $((G_{b_1} \setminus G_{b_2}) \cup (G_{b_2} \setminus G_{b_1}), E(G_{b_1} \setminus G_{b_2}, G_{b_2} \setminus G_{b_1}))$ we infer that for every pair of distinct $b_1, b_2 \in B_g$ there is at least one arc with endpoints b_1 and b_2. Hence S is semi-complete. The definition of S gives raise to a straightforward algorithm constructing it in $\mathcal{O}(p(k)^2 \cdot n^2)$ time.

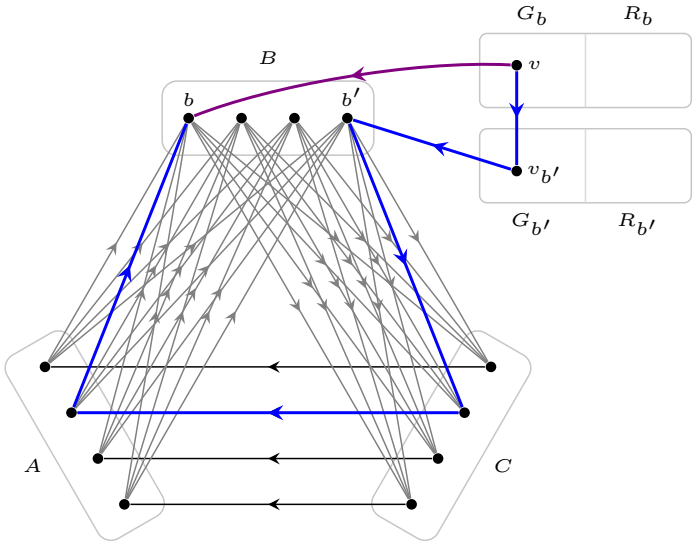

Figure 7.1: Rerouting strategy for a path accessing vertex b from G_b. The original path is depicted in violet, while the rerouted path is depicted in blue.

Let X be the set of vertices of B_g that have outdegree at least $6k^2 + 6k$ in S; note that X can be constructed in $\mathcal{O}(p(k)^2)$ time. Observe that $|X| \geq 16k^2 + 16k + 1$, since otherwise the sum of the outdegrees in S would be at most $(16k^2 + 16k)(|B_g| - 1) + (|B_g| - 16k^2 - 16k)(6k^2 + 6k - 1)$, which is smaller than $\binom{|B_g|}{2}$ for $|B_g| \geq 4(16k^2 + 16k + 1)$.

We now claim that for every $b \in X$, every path of $\mathcal{Q}$ using vertex b can be rerouted at the cost

of using at most two free vertices of B and at most two arcs from the matching that have still both endpoints free. We perform reroutings one by one. Assume that there is a path $P \in \mathcal{Q}$ accessing b from outside A. Let v be the predecessor of b on P. If $v \in R_b$, then we use the same rerouting strategy as in the case of large $B_\emptyset$. Assume then that $v \in G_b$. As $b \in X$, its outdegree in S is at least $6k^2 + 6k$. This means that there are at least $6k^2 + 6k$ vertices b' in B_g and corresponding vertices $v_{b'} \in N^-(b')$, such that for every b' either $v_{b'} = v$ or $(v, v_{b'}) \in E$. Out of these $6k^2+6k$ vertices b', at most $4k$ are not free due to Observation 103, at most $2k-2$ were used in previous reroutings, which leaves us with at least $6k^2$ vertices b' still being free. If for any such b' we have $v_{b'} = v$, we follow the same rerouting strategy as in the case of large $B_\emptyset$. Assume then that these $6k^2$ vertices $v_{b'}$ are all distinct from v; note that, however, they are not necessarily distinct from each other. As each $v_{b'}$ belongs to $G_{b'}$, $v_{b'}$ can have at most $6k-1$ outneighbors in B. Hence, each vertex of $V(T)$ can occur among these $6k^2$ vertices $v_{b'}$ at most $6k-1$ times, so we can distinguish at least $k+1$ pairwise distinct vertices $v_{b'}$. We have that arcs $(v, v_{b'})$ and $(v_{b'}, b')$ exist, while b' is still free. By Observation 99, at most k arcs $(v, v_{b'})$ are used by some paths, which leaves us at least one $v_{b'}$, for which arc $(v, v_{b'})$ is free. We can now substitute the arc (v, b) in P by the path $v \to v_{b'} \to b' \to c \to a \to v$, where (c, a) is an arbitrarily chosen arc from the matching that still has both endpoints free, which exists due to using at most $2k-2$ of them so far. See Figure 7.1. After rerouting, remove all cycles that appeared on the obtained walk in order to obtain a simple path; again, this shortcutting step does not spoil property (1.iii) for the path. Thus, Observation 99 still holds after rerouting. Observe that in this manner we use additional vertex b' that was free, additional one arc (c, a) from the matching, whereas passing the path through $v_{b'}$ can spoil at most one arc of the matching that still had both endpoints free, or at most one free vertex from B. This concludes the construction of the set X.

We proceed with the second step of the proof. We mimic the rerouting arguments from the first step to obtain a vertex $x \in X$ with the following property: the rerouted family of paths $\mathcal{P}$ obtained in the first step that can access x only from A, can be further rerouted so that every path can only leave x by accessing some vertex from C.

For every $b \in X$ consider sets R'_b and G'_b defined similarly as before:

$$R'_b = \{v \mid v \in V(T) \setminus C \wedge (b, v) \in E \wedge |N^-(v) \cap B| \geq 8k\},$$
$$G'_b = \{v \mid v \in V(T) \setminus C \wedge (b, v) \in E \wedge |N^-(v) \cap B| < 8k\}.$$

Assume first that there is some $y \in X$, such that $G'_y = \emptyset$. We argue that we can in such a case set $x = y$. Firstly, reroute a solution that minimizes the total sum of lengths of the paths obtaining a solution with the family of paths $\mathcal{P}$ that uses at most $2k$ additional free vertices from B and at most $2k$ additional arcs from the matching that had both endpoints free, but does not access x from outside A. One by one we reroute paths that traverse y. Each rerouting will cost at most one free vertex from B and at most one arc from the matching that has still both endpoints free. Let P be a path from the solution that passes through y and let $v \in R'_y$ be the next vertex on P. The vertex v has at least $8k$ inneighbors in B; at most $4k$ of them could be used by the original solution, at most $2k$ of them could be used in rerouting during the first phase and at most $k-1$ of them could be used during previous reroutings in this phase. Therefore, we are left with at least one vertex $y' \in B$ that is still free, such that $(y', v) \in E(T)$. We can now substitute the arc (y, v) in P by the path $y \to c \to a \to y' \to v$, where (c, a) is an arbitrarily chosen arc from the matching that was not yet used, which exists due to using at most $3k-1$ of them so far. Again, we shortcut all the cycles that appeared after this substitution so that we obtain a simple path. Note that this shortcutting step spoils neither property (2.ii) nor (2.iii) for the path.

We are left with the case when G'_y is nonempty for all $y \in X$. Construct a digraph $S' = (X, L')$ in symmetrically to the previous construction: put arc (b_1, b_2) into L' iff for every $v_{b_2} \in G'_{b_2}$ there exists $v_{b_1} \in G'_{b_1}$ such that $v_{b_1} = v_{b_2}$ or $(v_{b_1}, v_{b_2}) \in E$. The remark after Lemma 105 ensures that S' is semi-complete. Again, S' can be computed in $\mathcal{O}(p(k)^2 \cdot n^2)$ time.

As $|X| \geq 16k^2+16k+1$, there exists $x \in X$, which has indegree at least $8k^2+8k$ in S'; note that x can be found in $\mathcal{O}(p(k)^2)$ time. As before, we argue that after the first rerouting phase for x, we can additionally reroute the paths so that every path can leave x only into C. We reroute the paths one by one; each rerouting uses at most two free vertices from B and at most two arcs from the matching that still had both endpoints free. As the indegree of x in S' is at least $8k^2 + 8k$, we have at least $8k^2+8k$ vertices $x' \in X$ and corresponding $v_{x'} \in G'_{x'}$, such that $v_{x'} = v$ or $(v_{x'}, v) \in E$. At most $4k$ of them were used in $\mathcal{Q}$, at most $2k$ were used in the first phase of rerouting, and at most $2k-2$ of them were used in this phase of rerouting. This leaves at least $8k^2$ vertices x' which are still free. If for any of them we have $v_{x'} = v$, we can make the rerouting similarly as in the previous case: we substitute the arc (x, v) with the path $x \to c \to a \to x' \to v$, where (c, a) is an arbitrary arc of the matching that still has both endpoints free, which exists due to using at most $4k - 2$ of them so far. Assume then, that all vertices $v_{x'}$ are distinct from v; note that, however, they are not necessarily distinct from each other. As for every x' we have $v_{x'} \in G'_{x'}$, by the definition of $G'_{x'}$ the vertices $v_{x'}$ can have at most $8k-1$ inneighbors in X. This means that every vertex of $V(T)$ can occur among vertices $v_{x'}$ at most $8k-1$ times, which proves that there are at least $k+1$ pairwise distinct vertices among them. By Observation 99, for at most k of them the arc outgoing to v can be already used, which leaves us with a single vertex x' such that arcs $(x', v_{x'})$ and $(v_{x'}, v)$ exist and are not yet used, whereas x' is still free. Now we can perform the rerouting as follows: we substitute the arc (x, v) in P by the path $x \to c \to a \to x' \to v_{x'} \to v$, where (c, a) is an arbitrary arc from the matching that still had both endpoints free. Such an arc exists since we used at most $2k$ such arcs in the first phase of the rerouting, and at most $2k-2$ in this phase of the rerouting. Similarly as before, we use one additional free vertex x' from B, one additional arc (c, a) from the matching, while usage of $v_{x'}$ can spoil at most one free vertex from B or at most one arc from the matching. After this, we delete all the possible cycles created on the path in order to make Observation 99 still hold; again, this shortcutting step spoils neither property (2.ii) nor property (2.iii). This concludes the construction of the vertex x.

To finish the proof it remains to show that after performing the two phases of rerouting and obtaining a solution η', whose paths can access and leave x only from A and into C, we can reroute every path so that it does not traverse x at all. Note that so far we have used at most $4k$ vertices from B, so we still have at least $k + 1$ vertices unused. Observe that at most k of these vertices can belong to $\eta'(V(H))$, which leaves us with at least one vertex x' that is still free and is not an image of any vertex of H in η'.

If $x \in \eta(u)$ for some $u \in V(H)$, then we simply move the image u: we consider η'' that differs from η' by replacing x with x' both in the image of u and in all the paths from $\eta'(E(H))$ which traverse x. Note that we can make this move since x is not a root vertex: the triple does not contain any roots. In case when $x \notin \eta(V(H))$ we can perform the same rerouting scheme: in all the paths from $\eta'(E(H))$ we substitute every appearance of x with x'. Then no path traverses x, so $\mathcal{I}' = ((H, u_1, u_2, \ldots, u_t), (T \setminus \{x\}, v_1, v_2, \ldots, v_t))$ is a YES instance if $\mathcal{I}$ was. □

7.3.2 Applying the irrelevant vertex rule

Armed with the irrelevant vertex rule, we can proceed to the algorithm for Rooted Immersion.

Theorem 106. *There exists an algorithm that, given a rooted semi-complete digraph* $\mathbf{T}$ *on* n *vertices and a rooted digraph* $\mathbf{H}$*, in time* $f(|H|) \cdot n^3$ *checks whether there exists a rooted immersion from* $\mathbf{H}$ *to* $\mathbf{T}$*, for some elementary function* f*.*

Proof. Let f be the function given by Lemma 44; i.e., basing on a $f(t)$-jungle in any semi-complete digraph S, one can find a t-triple in S in time $\mathcal{O}(|V(S)|^3 \log |V(S)|)$. Moreover, let p be the polynomial given by Theorem 98; i.e., in a $p(|H|)$-triple that is disjoint from the roots one can find an irrelevant vertex for the Rooted Immersion problem in $\mathcal{O}(p(|H|)^2 \cdot n^2)$ time.

Given the input semi-complete rooted digraph T and a rooted digraph H, we run the approximation algorithm of Theorem 65 for parameters $5k$ and $130k$ on T with the roots removed, where $k = f(p(|H|))$; this takes at most $g(|H|) \cdot n^2$ time for some elementary function g. If the algorithm returns a decomposition of T without roots of width at most $140k$, we include all the roots in every bag of the decomposition and finalize the algorithm by running the dynamic programming routine for Rooted Immersion (Theorem 92), which takes $h(|H|) \cdot n$ time for some elementary function h. Otherwise, using Lemma 47 or Lemma 50 we extract a $(k, 4)$-short jungle X from the output $(130k + 2, 5k)$-degree tangle or $(5k + 1, 5k)$-matching tangle; this takes $\mathcal{O}(k^3 \cdot n^2)$ time.

Obviously, X is also a k-jungle in the sense of Fradkin and Seymour, so we are tempted to run the algorithm of Lemma 44 to extract a triple; however, the running time is a bit too much. We circumvent this obstacle in the following manner. As X is a $(k, 4)$-short jungle, then if we define S to be the subdigraph induced in T by X and, for every pair v, w of vertices in X, k internally vertex-disjoint paths of length at most 4 from v to w, then X is still a $(k, 4)$-short jungle in S, but S has size polynomial in k. As we store the short jungle together with the corresponding family of paths between the vertices, we can construct S in $\mathcal{O}(k^{\mathcal{O}(1)})$ time and, using Lemma 44, in $\mathcal{O}(k^{\mathcal{O}(1)})$ time find a $p(|H|)$-triple inside S. This $p(|H|)$-triple is of course also a $p(|H|)$-triple in T. We apply Theorem 98 to find an irrelevant vertex in this triple in $p(|H|)^2 \cdot n^2$ time, delete it, and restart the algorithm.

Since there are n vertices in the graph, the algorithm makes at most n iterations. Since every iteration takes $h(k) \cdot n^2$ time for some elementary function h, the claim follows. □

7.4 Dynamic programming routines for containment relations

In this section we provide details of the dynamic programming routines that solve containment problems when a path decomposition of small width is given. First, we explain the terminology used to describe the constructed parts of expansions or immersions. Then we explain the routine for topological containment that is somewhat simpler, and finally proceed to immersion. But before all of these, let us give some intuition behind the algorithms.

The main idea of our routine is as follows: we will encode the interaction of the model of H with all the already introduced vertices as sequences of paths in a standard folio manner. Every such path has to end in the separator, but can begin in any forgotten vertex since it may be accessed from a vertex not yet introduced. In general setting the dynamic program would need to remember this first vertex in order to check whether it can be indeed accessed; that would yield an XP algorithm and, in essence, this is exactly the idea behind the algorithm of Fradkin and Seymour [153]. However, if the digraph is semi-complete, then between every not yet introduced vertex and every forgotten vertex there is an arc. Therefore, we do not need to remember the forgotten vertex itself to check accessibility; a marker saying *forgotten*, together with information about which markers in fact represent the same vertices in case of immersion, will suffice.

We hope that a reader well-familiar with construction of dynamic programs on various decompositions already has a crude idea about how the computation will be performed. Let us proceed with the details in the next subsections.

7.4.1 Terminology

First, we need to introduce definitions that will enable us to encode all the possible interactions between a model of a digraph H and a separation. Let (A, B) be a separation of T, where T is a given semi-complete digraph.

In the definitions we use two special symbols: $\mathfrak{F}, \mathfrak{U}$; the reader can think of them as an arbitrary element of $A \setminus B$ (*forgotten*) and $B \setminus A$ (*unknown*), respectively. Let $\iota : V(T) \to (A \cap B) \cup \{\mathfrak{F}, \mathfrak{U}\}$ be defined as follows: $\iota(v) = v$ if $v \in A \cap B$, whereas $\iota(v) = \mathfrak{F}$ for $v \in A \setminus B$ and $\iota(v) = \mathfrak{U}$ for $v \in B \setminus A$.

Definition 107. *Let P be a path. A sequence of paths $(P_1, P_2, \ldots, P_h)$ is a* trace *of P with respect to (A, B), if P_i for $1 \leq i \leq h$ are all maximal subpaths of P that are fully contained in $T[A]$, and the indices in the sequence reflect their order on path P.*

Let $(P_1, P_2, \ldots, P_h)$ be the trace of P with respect to (A, B). A signature *of P on (A, B) is a sequence of pairs $((b_1, e_1), (b_2, e_2), \ldots, (b_h, e_h))$, where $b_h, e_h \in (A \cap B) \cup \{\mathfrak{F}\}$, such that for every $i \in \{1, 2, \ldots, h\}$:*

- *b_i is the beginning of path P_i if $b_i \in A \cap B$, and $\mathfrak{F}$ otherwise;*
- *e_i is the end of path P_i if $e_i \in A \cap B$, and $\mathfrak{F}$ otherwise.*

In other words, b_i, e_i are images of the beginning and the end of path P_i in mapping ι. Observe the following properties of the introduced notion:

- Signature of a path P on separation (A, B) depends only on its trace; therefore, we can also consider signatures of traces.
- It can happen that $b_i = e_i \neq \mathfrak{F}$ only if P_i consists of only one vertex $b_i = e_i$.
- From the definition of separation it follows that only for $i = h$ it can happen that $e_i = \mathfrak{F}$, since there is no arc from $A \setminus B$ to $B \setminus A$.
- The empty signature corresponds to P entirely contained in $B \setminus A$.

Now we are able to encode relevant information about a given expansion of H.

Definition 108. *Let η be an expansion of a digraph H in T. An* expansion signature *of η on (A, B) is a mapping ρ such that:*

- *for every $v \in V(H)$, $\rho(v) = \iota(\eta(v))$;*
- *for every $a \in E(H)$, $\rho(a)$ is a signature of $\eta(a)$ on (A, B).*

The set of possible expansion signatures on separation (A, B) will be denoted by $\mathcal{V}_{(A,B)}$. Observe that in an expansion η all the paths in $\eta(E(H))$ are internally vertex-disjoint, so the non-forgotten beginnings and ends in all the signatures of paths from $\rho(E(H))$ can be equal only if they are in the same pair or correspond to a vertex from $\rho(V(H))$ at the beginning or at the end of a signature of some path. Armed with this observation, we can bound the number of possible expansion signatures on a separation of small order.

Lemma 109. *If $|V(H)| = k, |E(H)| = \ell, |A \cap B| = m$, then the number of possible different expansion signatures on (A, B) is at most*

$$(m+2)^k \cdot (m+2)^m \cdot m^\ell \cdot m! \cdot (m+2)^\ell \cdot (m+2)^\ell = 2^{\mathcal{O}((k+\ell+m)\log m)}.$$

Moreover, all of them can be enumerated in $2^{\mathcal{O}((k+\ell+m)\log m)}$ time.

Proof. The consecutive terms correspond to:

1. the choice of mapping ρ on $V(H)$;
2. for every element of $(A \cap B) \setminus \rho(V(H))$, choice whether it will be the end of some subpath in some path signature, and in this case, the value of corresponding beginning (a vertex from $A \cap B$ or $\mathfrak{F}$);
3. for every pair composed in such manner, choice to which $\rho(a)$ it will belong;
4. the ordering of pairs along the path signatures;
5. for every $(v, w) \in E(H)$, choice whether to append a pair of form $(b, \rho(w))$ at the end of the signature $\rho((v, w))$, and in this case, the value of b (a vertex from $A \cap B$ or $\mathfrak{F}$).
6. for every $(v, w) \in E(H)$, choice whether to append a pair of form $(\rho(v), e)$ at the beginning of the signature $\rho((v, w))$, and in this case, the value of e (a vertex from $A \cap B$ or $\mathfrak{F}$).

It is easy to check that using all these information one can reconstruct the whole signature. For every object constructed in the manner above we can check in time polynomial in k, ℓ, m, whether it corresponds to a possible signature. This yields the enumeration algorithm. □

We now proceed to encoding intersection of an immersion with a given separation (A, B). Unfortunately, the definition must be slightly more complicated for the following reason. Assume that we have two subpaths P_1, P_2 in some path traces that both start in some vertices b_1, b_2 that are forgotten, i.e., $b_1, b_2 \in A \setminus B$. Observe that not only need to remember that $\iota(b_1) = \iota(b_2) = \mathfrak{F}$, but also need to store the information whether $b_1 = b_2$: in the future computation we might need to know whether b_1 and b_2 are actually not the same vertex, in order to prevent using twice the same arc incoming to this vertex from the unknown region, in two different images of paths. Fortunately, this is the only complication.

Definition 110. *Let η be an immersion of a digraph H in T. An* immersion signature *of η on (A, B) is a mapping ρ together with an equivalence relation $\equiv$ on the set of all the pairs of form $(\mathfrak{F}, e)$ appearing in the image of ρ, such that:*

- *for every $v \in V(H)$, $\rho(v) = \iota(\eta(v))$;*
- *for every $a \in E(H)$, $\rho(a)$ is a signature of $\eta(a)$;*
- *$(\mathfrak{F}, e_1) \equiv (\mathfrak{F}, e_2)$ if and only if markers $\mathfrak{F}$ in both pairs correspond to the same forgotten vertex before being mapped by ι.*

We remark that the same pair of form $(\mathfrak{F}, e)$ can appear in different signatures; in this case, by somehow abusing the notation, we consider all the appearances as different pairs. Also, we often treat the equivalence relation $\equiv$ as part of the mapping ρ, thus denoting the whole signature by ρ. We denote the set of possible immersion signatures on separation (A, B) by $\mathcal{E}_{(A,B)}$. Similarly as in Lemma 109, we can bound the number of immersion signatures on a separation of small order.

Lemma 111. *If $|V(H)| = k, |E(H)| = \ell, |A \cap B| = m$, then the number of possible different immersion signatures on (A, B) is bounded by*

$$(m+2)^k \cdot ((m+2)^m \cdot m! \cdot (m+2)^2)^\ell \cdot B_{(m+2)\ell} = 2^{\mathcal{O}(k \log m + m\ell(\log \ell + \log m))}.$$

Moreover, all of them can be enumerated in $2^{\mathcal{O}(k \log m + m\ell(\log \ell + \log m))}$ time.

Proof. The consecutive terms correspond to:

1. the choice of mapping ρ on $V(H)$;
2. for every arc $a = (v, w) \in E(H)$ the complete information about the signature $\rho(a)$:
   - for every element of $A \cap B$, whether it will be the end of some path in the signature, and in this case, the value of corresponding beginning (a vertex from $A \cap B$ or $\mathfrak{F}$),
   - the ordering of pairs along the signature,
   - whether to append a pair of form $(b, \rho(w))$ at the end of the signature $\rho(a)$, and in this case, the value of b (a vertex from $A \cap B$ or $\mathfrak{F}$),
   - whether to append a pair of form $(\rho(v), e)$ at the beginning of the signature $\rho(a)$, and in this case, the value of e (a vertex from $A \cap B$ or $\mathfrak{F}$).
3. partition of at most $(m+2)l$ pairs in all the signatures from $\rho(E(H))$ into equivalence classes with respect to $\equiv$.

In the last term we used Bell numbers B_n, for which a trivial bound $B_n \leq n^n$ applies.

It is easy to check that using all these information one can reconstruct the whole signature. For every object constructed in the manner above we can check in time polynomial in k, l, m, whether it corresponds to a possible signature. This yields the enumeration algorithm. □

7.4.2 The algorithm for topological containment

Finally, we are able to present the dynamic programming routine for topological containment. Recall that we will solve a slightly more general problem, where arcs from some subset $F \subseteq E(H)$, called constraints, are required to be mapped to single arcs instead of possibly longer paths. We first prove a simple lemma that will be useful to handle the constraints.

Lemma 112. *Let H, G be simple digraphs, and let $F \subseteq E(H)$. Then H can be topologically embedded in G with constraints F if and only if $H' = H \setminus F$ can be topologically embedded in G using expansion η such that $(\eta(v), \eta(w)) \in E(G)$ for every $(v, w) \in F$.*

Proof. From left to right, if η is an expansion of H in G that respects constraints F, then η restricted to H' is also an expansion of H' in G, and moreover $(\eta(v), \eta(w)) \in E(G)$ for every $(v, w) \in F$ since η respects constraints F. From right to left, assume that η is an expansion of

H' in G such that $(\eta(v), \eta(w)) \in E(G)$ for every $(v, w) \in F$. Then we can extend η to H by setting $\eta((v, w)) = (\eta(v), \eta(w))$ for every $(v, w) \in F$, and these new paths will be pairwise internally vertex-disjoint, and internally vertex-disjoint from $\eta(a)$ for $a \notin F$. □

We are ready to provide the dynamic programming routine.

Theorem 113 (Theorem 91, restated)**.** *There exists an algorithm that, given a digraph H with k vertices, ℓ arcs, a set $F \subseteq E(H)$ of constraints, and a semi-complete digraph T on n vertices together with its path decomposition of width p, checks whether H is topologically contained in T with constraints F in time $2^{\mathcal{O}((p+k+\ell)\log p)} \cdot n$.*

Proof. Let $H' = H \setminus F$, and let $W = (W_1, \ldots, W_r)$ be a path decomposition of T of width p. Without loss of generality we assume that W is a nice path decomposition and $r = \mathcal{O}(n)$.

By Lemma 109, for every separation $(A, B) = (\bigcup_{j=1}^{i} W_j, \bigcup_{j=i}^{r} W_j)$ with separator W_i the number of possible signatures is $2^{\mathcal{O}((p+k+\ell)\log p)}$. We will consecutively compute the values of a binary table $D_{(A,B)} : \mathcal{V}_{(A,B)} \to \{\bot, \top\}$ with the following meaning. For $\rho \in \mathcal{V}_{(A,B)}$, $D_{(A,B)}[\rho]$ tells whether there exists a mapping $\overline{\rho}$ with the following properties:

- for every $v \in V(H)$, $\overline{\rho}(v) = \rho(v)$ if $\rho(v) \in (A \cap B) \cup \{\mathfrak{U}\}$ and $\overline{\rho}(v) \in A \setminus B$ if $\rho(v) = \mathfrak{F}$;
- for every $a = (v, w) \in E(H')$, $\overline{\rho}(a)$ is a correct path trace with signature $\rho(a)$, beginning in $\overline{\rho}(v)$ if $\overline{\rho}(v) \in A$ and anywhere in A otherwise, ending in $\overline{\rho}(w)$ if $\overline{\rho}(w) \in A$ and anywhere in $A \cap B$ otherwise;
- path traces $\overline{\rho}(a)$ are vertex-disjoint, apart possibly from meeting at the ends if the ends correspond to images of appropriate endpoints of arcs in $\overline{\rho}$;
- for every $a = (v, w) \in F$, if $\overline{\rho}(v) \neq \mathfrak{U}$ and $\overline{\rho}(w) \neq \mathfrak{U}$, then $(\overline{\rho}(v), \overline{\rho}(w)) \in E(T)$.

Such mapping $\overline{\rho}$ will be called a *partial expansion of H' on (A, B) respecting constraints F*. Note that we may also talk about signatures of partial expansions; in the definition above, ρ is the signature of $\overline{\rho}$ on (A, B).

For the first separation $(\emptyset, V(T))$ we have exactly one signature with value $\top$, being the signature which maps all the vertices into $\mathfrak{U}$ and all the arcs into empty signatures. By Lemma 112, the result of the whole computation should be the value for the signature on the last separation $(V(T), \emptyset)$ which maps all vertices into $\mathfrak{F}$ and arcs into signatures consisting of one pair $(\mathfrak{F}, \mathfrak{F})$. Therefore, it suffices to show how to fill the values of the table for **introduce vertex** step and **forget vertex** step. Note that the introduce bags of the decomposition may be viewed as introducing a vertex v to a separation (A, B), i.e., considering the next separation $(A \cup \{v\}, B)$ for $v \notin A$, $v \in B$. Similarly, forget bags may be viewed as forgetting a vertex w from a separation $(A, B \cup \{w\})$ with $w \in A$, $w \notin B$, i.e., considering the next separation (A, B).

Introduce vertex step. Let us introduce vertex $v \in B \setminus A$ to the separation (A, B), i.e., we consider the new separation $(A \cup \{v\}, B)$. Let $\rho \in \mathcal{V}_{(A\cup\{v\},B)}$. We show that $D_{(A\cup\{v\},B)}[\rho]$ can be computed using the stored values of $D_{(A,B)}$ by analyzing the way signature ρ interferes with vertex v. In each case we argue that one can take $D_{(A\cup\{v\},B)}[\rho] = \bigvee_{\rho' \in \mathcal{G}} D_{(A,B)}[\rho']$ for some set $\mathcal{G}$ that corresponds to possible trimmings of ρ to the previous separation of smaller order. Formally, one needs to argue that (i) if there exists a partial expansion with signature ρ on $(A \cup \{v\}, B)$, then after

deleting vertex v this partial expansion has some signature $\rho' \in \mathcal{G}$ on (A, B), and (ii) if there exists a partial expansion with signature ρ' on (A, B) such that $\rho' \in \mathcal{G}$, then one can extend this partial expansion using vertex v to obtain a partial expansion on $(A \cup \{v\}, B)$ with signature ρ. Since this check in each case follows directly from the explanations provided along with the construction of $\mathcal{G}$, we leave the formal verification of this statement to the reader.

Case 1*:* $v \notin \rho(V(H))$, that is, v is not an image of a vertex of H. Observe that in this situation v can be contained in at most one pair of at most one signature $\rho(a)$ for some $a \in E(H')$.

Case 1.1*:* $b_i = v = e_i$ for some pair $(b_i, e_i) \in \rho(a)$ and some $a \in E(H')$. This means that the signature of the partial expansion truncated to separation (A, B) must look exactly like ρ, but without this subpath of length zero. Thus $D_{(A\cup\{v\},B)}[\rho] = D_{(A,B)}[\rho']$, where ρ' is constructed from ρ by deleting this pair from the corresponding signature.

Case 1.2*:* $b_i = v \neq e_i$ for some pair $(b_i, e_i) \in \rho(a)$ and some $a \in E(H')$. This means that the partial expansion truncated to separation (A, B) has to look the same but for the path corresponding to this very pair, which needs to be truncated by vertex v. The new beginning has to be either a vertex in $A \cap B$, or a forgotten vertex from $A \setminus B$. As T is semi-complete and (A, B) is a separation, there is an arc from v to every vertex of $A \setminus B$. Therefore, $D_{(A\cup\{v\},B)}[\rho] = \bigvee_{\rho'} D_{(A,B)}[\rho']$, where the disjunction is taken over all signatures ρ' differing from ρ as follows: in ρ' the pair (b_i, e_i) is substituted with (b'_i, e_i), where $b'_i = \mathfrak{F}$ or b'_i is any vertex of $A \cap B$ such that there is an arc (v, b'_i).

Case 1.3*:* $b_i \neq v = e_i$ for some pair $(b_i, e_i) \in \rho(a)$ and some $a \in E(H')$. Similarly as before, the partial expansion truncated to separation (A, B) has to look the same but for the path corresponding to this very pair, which needs to be truncated by vertex v. As (A, B) is a separation, the previous vertex on the path has to be in the separator $A \cap B$. Therefore, $D_{(A\cup\{v\},B)}[\rho] = \bigvee_{\rho'} D_{(A,B)}[\rho']$, where the disjunction is taken over all signatures ρ' differing from ρ as follows: in ρ' the pair (b_i, e_i) is substituted with (b_i, e'_i), where e'_i is any vertex of $A \cap B$ such that there is an arc (e'_i, v).

Case 1.4*:* v is not contained in any pair in any signature from $\rho(E(H'))$. Either v lies on some path in the partial expansion, or it does not. In the first case the corresponding path in partial expansion on $(A \cup \{v\}, B)$ has to be split into two subpaths, when truncating the expansion to (A, B). Along this path, the arc that was used to access v had to come from inside $A \cap B$, due to (A, B) being a separation; however, the arc used to leave v can go to $A \cap B$ or to any forgotten vertex from $A \setminus B$, as (A, B) is a separation and T is a semi-complete digraph. In the second case, the signature of the truncated expansion stays the same. Therefore, $D_{(A\cup\{v\},B)}[\rho] = D_{(A,B)}[\rho] \vee \bigvee_{\rho'} D_{(A,B)}[\rho']$, where the disjunction is taken over all signatures ρ' differing from ρ as follows: in ρ' exactly one pair (b_i, e_i) is substituted with two pairs (b_i, e'_i) and (b'_i, e_i), where $e'_i \in A \cap B$ with $(e'_i, v) \in E(T)$, whereas $b'_i = \mathfrak{F}$ or $b'_i \in A \cap B$ with $(v, b'_i) \in E(T)$.

Case 2*:* $v = \rho(u)$ for some $u \in V(H)$. For every $(u, u') \in E(H')$, v has to be the beginning of the first pair of $\rho((u, u'))$; otherwise, $D_{(A\cup\{v\},B)}[\rho] = \bot$. Similarly, for every $(u', u) \in E(H')$, v has to be the end of the last pair of $\rho((u', u))$; otherwise, $D_{(A\cup\{v\},B)}[\rho] = \bot$. Furthermore, for every $(u, u') \in F$ such that $\rho(u') \neq \mathfrak{U}$, if $\rho(u') \in A \cap B$ then $(\rho(u), \rho(u'))$ must be an arc of T; otherwise, $D_{(A\cup\{v\},B)}[\rho] = \bot$. Finally, for every $(u', u) \in F$ such that $\rho(u') \neq \mathfrak{U}$, it must hold that $\rho(u') \in A \cap B$ and $(\rho(u'), \rho(u))$ must be an arc of T; otherwise, $D_{(A\cup\{v\},B)}[\rho] = \bot$.

Assume then that all these four assertions hold. Then $D_{(A\cup\{v\},B)}[\rho] = \bigvee_{\rho'} D_{(A,B)}[\rho']$, where the disjunction is taken over all signatures ρ' such that: (i) ρ' differs on $V(H)$ from ρ only by having $\rho'(u) = \mathfrak{U}$; (ii) the first pairs of all $\rho((u,u'))$ are truncated as in Case 1.2 for all $(u,u') \in E(H)$ (or as in Case 1.1, if the beginning and the end coincide), and (iii) the last pairs of all $\rho((u',u))$ are truncated as in Case 1.3 for all $(u',u) \in E(H)$ (or as in Case 1.1, if the beginning and the end coincide).

Forget vertex step Let us forget vertex $w \in A \setminus B$ from the separation $(A, B\cup\{w\})$, i.e., we consider the new separation (A,B). Let $\rho \in \mathcal{V}_{(A,B)}$; we argue that $D_{(A,B)}[\rho] = \bigvee_{\rho'\in\mathcal{G}} D_{(A\cup\{w\},B)}[\rho']$ for some set $\mathcal{G} \subseteq \mathcal{V}_{(A\cup\{w\},B)}$, which corresponds to possible extensions of ρ to the previous separation of larger order. Formally, one needs to argue that (i) if there exists a partial expansion with signature ρ on (A,B), then this partial expansion has signature $\rho' \in \mathcal{G}$ on $(A, B\cup\{w\})$, and (ii) if there exists a partial expansion with signature ρ' on $(A, B\cup\{w\})$ such that $\rho' \in \mathcal{G}$, then the signature of this partial expansion on (A,B) is ρ. Since this check in each case follows directly from the explanations provided along with the construction of $\mathcal{G}$, we leave the formal verification of this statement to the reader.

We now discuss, which signatures ρ' are needed in $\mathcal{G}$ by considering all the signatures $\rho' \in \mathcal{V}_{(A\cup\{w\},B)}$ partitioned with respect to behaviour on vertex w. For a fixed signature ρ' we put constraints on how ρ' must look like to be included in $\mathcal{G}$: we first put a constraint on the image of $V(H)$ in ρ', and then for every $a \in E(H')$ we list the possible values of $\rho'(a)$. In $\mathcal{G}$ we take into the account all signatures ρ' that (i) satisfy the imposed constraints on the image of $V(H)$, (ii) have one of the correct values for every $a \in E(H')$.

Case 1*:* $w \notin \rho'(V(H))$, that is, w is not in the image of $V(H)$. In this case we require that $\rho'|_{V(H)} = \rho|_{V(H)}$. We now inspect one arc $a \in E(H')$ and determine the correct values of $\rho'(a)$ by looking at all possible values of $\rho'(a)$ partitioned with respect to behaviour on w.

Case 1.1*:* $b_i = w = e_i$ for some pair $(b_i, e_i) \in \rho'(a)$. This means that in the corresponding partial expansions w had to be left to $B \setminus A$; however, in T there is no arc from w to $B \setminus A$ since (A,B) is a separation. Therefore, in $\mathcal{G}$ we consider no signatures ρ' having such a behaviour on any arc a.

Case 1.2*:* $b_i = w \neq e_i$ for some pair $(b_i, e_i) \in \rho'(a)$. If we are to consider ρ' in $\mathcal{G}$, then w must prolong some path from the signature ρ in such a manner that w is its beginning. After forgetting w the beginning of this path belongs to the forgotten vertices; therefore, in $\mathcal{G}$ we consider only signatures with $\rho'(a)$ differing from $\rho(a)$ on exactly one pair: in $\rho'(a)$ there is (w, e_i) instead of $(\mathfrak{F}, e_i)$ in $\rho(a)$.

Case 1.3*:* $b_i \neq w = e_i$ for some pair $(b_i, e_i) \in \rho'(a)$. This means that in the corresponding partial expansions w have to be left to $B \setminus A$; however, in T there is no arc from w to $B \setminus A$ since (A,B) is a separation. We obtain a contradiction; therefore, in $\mathcal{G}$ we consider no signatures having such a behaviour on any arc a.

Case 1.4*:* w is not contained in any pair of $\rho'(a)$. In this case, in the corresponding partial expansions w has to be either unused by the trace of a, or be an internal vertex of a path in this trace. In both cases the signatures on $(A, B\cup\{w\})$ and on (A,B) are equal. It follows that in $\mathcal{G}$ we can consider only signatures with $\rho'(a) = \rho(a)$ for such arcs a.

Case 2: $w = \rho'(u)$ for some $u \in V(H)$. We consider in $\mathcal{G}$ signatures ρ' that differ from ρ in following manner: (i) ρ' differs on $V(H)$ from ρ only by having $\rho'(u) = w$ for exactly one vertex $u \in V(H)$ whereas $\rho(u) = \mathfrak{F}$; (ii) for all arcs $(u, u') \in E(H')$, the first pair of $\rho'((u, u'))$ is of form (w, e_1), whereas the first pair of $\rho((u, u'))$ is of form $(\mathfrak{F}, e_1)$; (iii) for all arcs $(u', u) \in E(H')$, the last pair of $\rho'((u', u))$ is of form (b_h, w), whereas the last pair of $\rho((u', u))$ is of form $(b_h, \mathfrak{F})$ (or $(\mathfrak{F}, \mathfrak{F})$ in case $b_h = w$).

Updating table $D_{(A,B)}$ for each separation requires at most $\mathcal{O}(|\mathcal{V}_{(A,B)}|^2 \cdot (p + k + \ell)^{\mathcal{O}(1)}) = 2^{\mathcal{O}((p+k+\ell)\log p)}$ time, and since the number of separations in the path decomposition W is $\mathcal{O}(n)$, the theorem follows. □

7.4.3 The algorithm for Rooted Immersion

Theorem 114 (Theorem 92, restated)**.** *There exists an algorithm that, given a rooted digraph $(H; v_1, v_2, \ldots, v_t)$ with k vertices and ℓ arcs and a semi-complete rooted digraph $(T; w_1, w_2, \ldots, w_t)$ on n vertices together with its path decomposition of width p, checks whether H can be immersed into T while preserving roots in time $2^{\mathcal{O}(k \log p + p\ell(\log \ell + \log p))} \cdot n$.*

Proof. Let $W = (W_1, \ldots, W_r)$ be the given path decomposition of T of width p. Without loss of generality we assume that W is a nice path decomposition and $r = \mathcal{O}(n)$.

By Lemma 111, for every separation $(A, B) = (\bigcup_{j=1}^{i} W_j, \bigcup_{j=i}^{r} W_j)$ with separator W_i we can bound the number of possible signatures by $2^{\mathcal{O}(k \log p + p\ell(\log \ell + \log p))}$. We show how to compute the values of a binary table $D_{(A,B)} : \mathcal{E}_{(A,B)} \to \{\bot, \top\}$ with the following meaning. For $\rho \in \mathcal{E}_{(A,B)}$, $D_{(A,B)}[\rho]$ tells, whether there exists a mapping $\overline{\rho}$ with the following properties:

- for every $v \in V(H)$, $\overline{\rho}(v) = \rho(v)$ if $\rho(v) \in (A \cap B) \cup \{\mathfrak{U}\}$ and $\overline{\rho}(v) \in A \setminus B$ if $\rho(v) = \mathfrak{F}$;
- for every $i = 1, 2, \ldots, t$, $\overline{\rho}(v_i) = w_i$ if $w_i \in A$ and $\overline{\rho}(v_i) = \mathfrak{U}$ otherwise;
- for every $a = (v, w) \in E(H)$, $\overline{\rho}(a)$ is a correct path trace with signature $\rho(a)$, beginning in $\overline{\rho}(v)$ if $\overline{\rho}(v) \in A$ and anywhere in A otherwise, ending in $\overline{\rho}(w)$ if $\overline{\rho}(w) \in A$ and anywhere in $A \cap B$ otherwise;
- all the paths in path traces $\overline{\rho}(a)$ are arc-disjoint for $a \in E(H)$.

Such mapping $\overline{\rho}$ will be called a *partial immersion of H on (A, B)*. Note that we may also talk about signatures of partial immersions; in the definition above, ρ is the signature of $\overline{\rho}$ on (A, B).

For the first separation $(\emptyset, V(T))$ we have exactly one signature with value $\top$, being the signature which maps all the vertices into $\mathfrak{U}$ and all the arcs into empty signatures. The result of the whole computation should be the value for the signature on the last separation $(V(T), \emptyset)$, which maps all the vertices to $\mathfrak{F}$ and arcs to signatures consisting of one pair $(\mathfrak{F}, \mathfrak{F})$. Therefore, it suffices to show how to fill the values of the table for **introduce vertex** step and **forget vertex** step. Similarly as in Theorem 113, we view these steps as introducing and forgetting a vertex from a separation.

Introduce vertex step Let us introduce vertex $v \in B \setminus A$ to the separation (A, B), i.e., we consider the new separation $(A \cup \{v\}, B)$. Let $\rho \in \mathcal{E}_{(A \cup \{v\}, B)}$, we need to show how to compute $D_{(A \cup \{v\}, B)}[\rho]$ by a careful case study of how the signature ρ interferes with vertex v. If $v = w_i$ for some $i \in \{1, 2, \ldots, t\}$, then we consider only such ρ for which $\rho(v_i) = v$; for all the others we fill false

values. Let us fix one $\rho \in \mathcal{E}_{(A\cup\{v\},B)}$. We argue that $D_{(A\cup\{v\},B)}[\rho] = \bigvee_{\rho'\in\mathcal{G}} D_{(A,B)}[\rho']$ for some set $\mathcal{G}$ that corresponds to possible trimmings of ρ to the previous separation of smaller order. Formally, one needs to argue that (i) if there exists a partial immersion with signature ρ on $(A \cup \{v\}, B)$, then after deleting vertex v this partial immersion has some signature $\rho' \in \mathcal{G}$ on (A, B), and (ii) if there exists a partial immersion with signature ρ' on (A, B) such that $\rho' \in \mathcal{G}$, then one can extend this partial immersion using vertex v to obtain a partial immersion on $(A \cup \{v\}, B)$ with signature ρ. Since this check in each case follows directly from the explanations provided along with the construction of $\mathcal{G}$, we leave the formal verification of this statement to the reader.

We examine every signature $\rho' \in \mathcal{E}_{(A,B)}$ and put constraints on how ρ' must look like to be included in $\mathcal{G}$: we first put a constraint on the image of $V(H)$ in ρ', and then for every $a \in E(H)$ we list the possible values of $\rho'(a)$. The set $\mathcal{G}$ consists of all the signatures ρ' that (i) satisfy the imposed constraints on the image of $V(H)$, (ii) have one of the correct values for every $a \in E(H)$, and (iii) satisfy some additional, global constraints that are described later.

Case 1*:* $v \notin \rho(V(H))$, that is, v is not being mapped on by any vertex of H. In this case we require that $\rho'|_{V(H)} = \rho|_{V(H)}$. We now examine one arc $a \in E(H)$ and list the correct values of $\rho'(a)$.

Case 1.1*:* $b_i = v = e_i$ for some pair $(b_i, e_i) \in \rho(a)$. This means that the signature$\rho(a)$ truncated to separation (A, B) must look exactly like $\rho(a)$, but without this subpath of length one. Thus we have one possible value for $\rho'(a)$, being $\rho(a)$ with this pair deleted.

Case 1.2*:* $b_i = v \neq e_i$ for some pair $(b_i, e_i) \in \rho(a)$. This means that the signature $\rho(a)$ truncated to separation (A, B) has to look the same as $\rho(a)$ but for the path corresponding to this very pair, which needs to be truncated by vertex v. The new beginning has to be either a vertex in $A \cap B$, or a forgotten vertex from $A \setminus B$. As T is semi-complete and (A, B) is a separation, there is an arc from v to every vertex of $A \setminus B$. Therefore, in $\rho'(a)$ the pair (b_i, e_i) has to be replaced with (b'_i, e_i), where $b'_i = \mathfrak{F}$ or b'_i is any vertex of $A \cap B$ such that there is an arc (v, b'_i). All the vertices $b'_i \neq \mathfrak{F}$ obtained in this manner have to be pairwise different. Moreover, we impose a condition that for all $a \in E(H)$ for which some pair of form (v, e_i) has been truncated to $(\mathfrak{F}, e_i)$, these pairs have to be pairwise non-equivalent with respect to $\equiv$ in ρ'; in this manner we forbid multiple usage of arcs going from v to the forgotten vertices.

Case 1.3*:* $b_i \neq v = e_i$ for some pair $(b_i, e_i) \in \rho(a)$. Similarly as before, signature $\rho(a)$ truncated to separation (A, B) has to look the same as $\rho(a)$ but for the path corresponding to this very pair, which needs to be truncated by vertex v. As (A, B) is a separation, the previous vertex on the path has to be in the separator $A \cap B$. Therefore, in $\mathcal{G}$ we take into consideration all signatures ρ' such that in ρ' the pair (b_i, e_i) is replaced with (b_i, e'_i), where e'_i is any vertex of $A \cap B$ such that there is an arc (e'_i, v). Also, all the vertices e'_i used in this manner for all $a \in E(H)$ have to be pairwise different.

Case 1.4*:* v is not contained in any pair in $\rho(a)$. Either v lies in the interior of some subpath from $\overline{\rho}(a)$, or it is not used by $\overline{\rho}(a)$ at all. In the first case the corresponding path in partial immersion on $(A\cup\{v\}, B)$ has to be split into two subpaths when truncating the immersion to (A, B). Along this path, the arc that was used to access v had to come from inside $A \cap B$, due to $(A \cup B)$ being a separation. However, the arc used to leave v can go to $A\cap B$ as well as to any forgotten vertex from $A\setminus B$, since (A, B) is a separation and T is a semi-complete digraph. In the second case, the signature of the truncated immersion stays the same. Therefore, in $\mathcal{G}$ we take into consideration signatures ρ' such

that they not differ from ρ on a, or in $\rho'(a)$ exactly one pair (b_i, e_i) is replaced with two pairs (b_i, e'_i) and (b'_i, e_i), where $e'_i \in A \cap B$ with $(e'_i, v) \in E(T)$, whereas $b'_i = \mathfrak{F}$ or $b'_i \in A \cap B$ with $(v, b'_i) \in E(T)$. Similarly as before, all vertices $b'_i \neq \mathfrak{F}$ used have to be pairwise different and different from those used in Case 1.2, all vertices e'_i used have to be pairwise different and different from those used in Case 1.3, and all the pairs $(\mathfrak{F}, e_i)$ created in this manner have to be pairwise non-equivalent and non-equivalent to those created in Case 1.2 (with respect to $\equiv$ in ρ').

Case 2: $v = \rho(u)$ for some $u \in V(H)$. For every $(u, u') \in E(H)$, v has to be the beginning of the first pair of $\rho((u, u'))$; otherwise, $D_{(A \cup \{v\}, B)} = \bot$. Similarly, for every $(u', u) \in E(H)$, v has to be the end of the last pair of $\rho((u', u))$; otherwise, $D_{(A \cup \{v\}, B)}[\rho] = \bot$. Assuming both of these assertions hold, into $\mathcal{G}$ we can take all signatures ρ' such that: (i) ρ' differs on $V(H)$ from ρ only by having $\rho'(u) = \mathfrak{U}$; and (ii) the images from $\rho'(E(H))$ follow exactly the same rules as in Cases 1.1-1.4.

Forget vertex step Let us forget vertex $w \in A \setminus B$ from the separation $(A, B \cup \{w\})$, i.e., we consider the new separation (A, B). Let $\rho \in \mathcal{E}_{(A,B)}$; we argue that $D_{(A,B)}[\rho] = \bigvee_{\rho' \in \mathcal{G}} D_{(A \cup \{w\}, B)}[\rho']$ for some set $\mathcal{G} \subseteq \mathcal{E}_{(A \cup \{w\}, B)}$, which corresponds to possible extensions of ρ to the previous separation of larger order. Formally, one needs to argue that (i) if there exists a partial immersion with signature ρ on (A, B), then this partial immersion has signature $\rho' \in \mathcal{G}$ on $(A, B \cup \{w\})$, and (ii) if there exists a partial immersion with signature ρ' on $(A, B \cup \{w\})$ such that $\rho' \in \mathcal{G}$, then the signature of this partial immersion on (A, B) is ρ. Since this check in each case follows directly from the explanations provided along with the construction of $\mathcal{G}$, we leave the formal verification of this statement to the reader.

We now discuss which signatures ρ' are needed in $\mathcal{G}$ by considering all the signatures $\rho' \in \mathcal{E}_{(A \cup \{w\}, B)}$ partitioned with respect to behaviour of the vertex w. For a fixed signature ρ' we put constraints on how ρ' must look like to be included in $\mathcal{G}$: we first put a constraint on the image of $V(H)$ in ρ', and then for every $a \in E(H)$ we list the possible values of $\rho'(a)$. In $\mathcal{G}$ we take into the account all signatures ρ' that (i) satisfy the imposed constraints on the image of $V(H)$, (ii) have one of the correct values for every $a \in E(H)$, and (iii) satisfy some additional, global constraints that are described later.

Case 1: $w \notin \rho'(V(H))$, that is, w is not an image of a vertex of H. In this case we require that $\rho'|_{V(H)} = \rho|_{V(H)}$. We now examine one arc $a \in E(H)$ and list the correct values of $\rho'(a)$ by looking at all possible values of $\rho'(a)$ partitioned with respect to behaviour on w.

Case 1.1: $b_i = w = e_i$ for some pair $(b_i, e_i) \in \rho'(a)$. This means that in the corresponding partial immersions w had to be left to $B \setminus A$; however, in T there is no arc from w to $B \setminus A$ as (A, B) is a separation. Therefore, in $\mathcal{G}$ we consider no signatures ρ' behaving in this manner on any arc a.

Case 1.2: $b_i = w \neq e_i$ for some pair $(b_i, e_i) \in \rho'(a)$. If we are to consider ρ' in $\mathcal{G}$, then w must prolong some path from the signature ρ in such a manner, that w is its beginning. After forgetting w the beginning of this path belongs to the forgotten vertices; therefore, in $\mathcal{G}$ we consider only signatures ρ' in which $\rho'(a)$ differs from $\rho(a)$ on exactly one pair: in $\rho'(a)$ there is (w, e_i) instead of $(\mathfrak{F}, e_i)$ in ρ. Moreover, all such pairs $(\mathfrak{F}, e_i)$ that were extended by w have to form one whole equivalence class with respect to $\equiv$ in ρ.

Case 1.3*:* $b_i \neq w = e_i$ for some pair $(b_i, e_i) \in \rho'(a)$. This means that in the corresponding partial expansions w have to be left to $B \setminus A$; however, in T there is no arc from w to $B \setminus A$ since (A, B) is a separation. We obtain a contradiction; therefore, in $\mathcal{G}$ we consider no signatures ρ' behaving in this manner on any arc a.

Case 1.4*:* w is not contained in any pair in $\rho'(a)$. In this case, in the corresponding partial expansions w has to be either unused by the trace of a, or be an internal vertex of a path in this trace. In both cases the signatures on $(A, B \cup \{w\})$ and on (A, B) are equal. It follows that in $\mathcal{G}$ we can consider only signatures with $\rho'(a) = \rho(a)$ for such arcs a.

Case 2*:* $w = \rho'(u)$ for some $u \in V(H)$. We consider in $\mathcal{G}$ signatures ρ' that differ from ρ in following manner: (i) ρ' differs on $V(H)$ from ρ only by having $\rho'(u) = w$ for exactly one vertex $u \in V(H)$ whereas $\rho(u) = \mathfrak{F}$; (ii) for all arcs $(u, u') \in E(H)$ the first pair of $\rho'((u, u'))$ is of form (w, e_1), whereas the first pair of $\rho((u, u'))$ is of form $(\mathfrak{F}, e_1)$; (iii) for all arcs $(u', u) \in E(H)$ the last pair of $\rho'((u', u))$ is of form (b_h, w), whereas the last pair of $\rho((u', u))$ is of form $(b_h, \mathfrak{F})$ (or $(\mathfrak{F}, \mathfrak{F})$ in case $b_h = w$); (iv) for all arcs $a \in E(H)$ non-incident with u we follow the same truncation rules as in Cases 1.1-1.4. Moreover, all the pairs in which w has been replaced with $\mathfrak{F}$ marker have to form one whole equivalence class with respect to $\equiv$.

Since updating the table $D_{(A,B)}$ for every separation (A, B) requires at most $\mathcal{O}(|\mathcal{E}_{(A,B)}|^2) = 2^{\mathcal{O}(k \log p + p\ell(\log \ell + \log p))}$ steps, while the number of separations in the pathwidth decomposition is $\mathcal{O}(n)$, the theorem follows. □

7.5 Conclusions and open problems

In this chapter we have presented how the approximation algorithm for pathwidth can be used to design FPT algorithms for testing various containment relations, and applied these algorithms to obtain some exemplary meta-results. We have also shown how to design an irrelevant vertex rule on a triple for rerouting a family of arc-disjoint paths, which leads to an FPT algorithm for the Rooted Immersion problem. It is noteworthy that the presented algorithms have much better running time guarantees than their famous analogues in the Graph Minors series of Robertson and Seymour. For instance, the celebrated algorithm testing whether H is a minor of G [285] runs in time $f(|H|) \cdot |V(G)|^3$ for a gargantuan function f that is not even specified; in particular, f is not elementary. The algorithms presented in this chapter have good dependency both on the size of the digraph to be embedded (single exponential), and on the size of the semi-complete digraph into which we embed (linear). This is mostly thanks to the usage of the new set of obstacles, especially the short jungles.

The first natural question stemming from the work of this chapter is whether this new set of obstacles, and in particular the short jungles, can give raise to more powerful irrelevant vertex rules. For example, if we consider the Rooted Immersion problem, it is tempting to try to replace finding an irrelevant vertex in a triple with a direct irrelevant vertex rule on a short jungle of size polynomial in the size of the digraph to be immersed. If this was possible, the running time of the algorithm for Rooted Immersion could be trimmed to single-exponential in terms of the size of the digraph to be immersed.

Perhaps a much more important question is whether one can also design an FPT algorithm for the rooted variant of topological containment testing, called further Rooted Topological Containment, since this was possible for Rooted Immersion. Observe that the classical Vertex Disjoint Paths is in fact a special case of Rooted Topological Containment where all the

vertices of H have specified images. Chudnovsky, Scott and Seymour [73] proved that k-Vertex Disjoint Paths is in class XP on semi-complete digraphs using a different approach. Their results also imply that the Rooted Topological Containment is in XP on semi-complete graphs. To the best of author's knowledge, the question whether Vertex Disjoint Paths in semi-complete digraphs can be solved in FPT time is still open.

Unfortunately, our approach used for Rooted Immersion does not apply directly to this problem. Dynamic programming on path decomposition works fine but the problem is with the irrelevant vertex arguments. Even for $k = 2$ there exist tournaments that contain arbitrarily large triples, but in which every vertex is relevant; let us now provide such an example.

For even n we construct a tournament T_n with two pairs of vertices (s_1, t_1), (s_2, t_2) so that the following properties are satisfied:

(i) T_n contains an $(n/2 - 1)$-triple;

(ii) there is exactly one solution to Vertex Disjoint Paths instance $(T_n, \{(s_1, t_1), (s_2, t_2)\})$ in which all the vertices of $V(T_n)$ lie on one of the paths.

This example shows that even though a graph can be complicated from the point of view of path decompositions, all the vertices can be relevant.

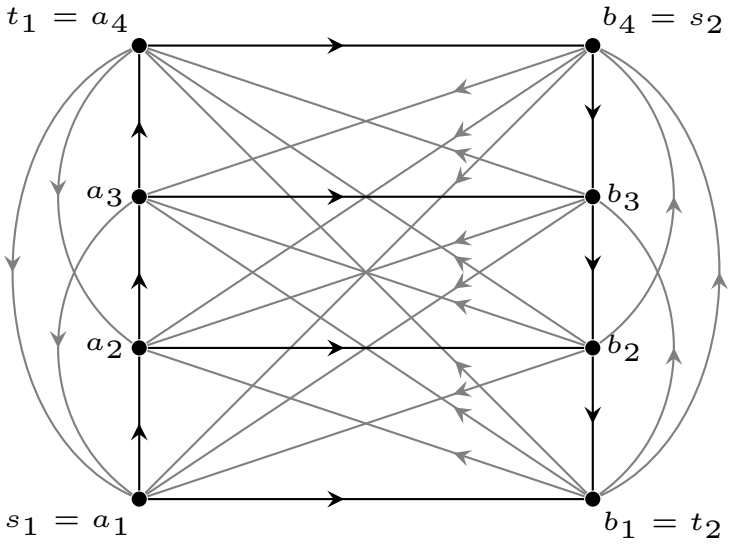

Figure 7.2: Tournament T_4.

Let $V(T_n) = \{a_i, b_i : 1 \leq i \leq n\}$, where $s_1 = a_1$, $t_1 = a_n$, $s_2 = b_n$ and $t_2 = b_1$. Construct following arcs:

- for every $i \in \{1, 2, \ldots, n-1\}$ create an arc (a_i, a_{i+1});
- for every $i, j \in \{1, 2, \ldots, n\}, i < j - 1$ create an arc (a_j, a_i);
- for every $i \in \{1, 2, \ldots, n-1\}$ create an arc (b_{i+1}, b_i);
- for every $i, j \in \{1, 2, \ldots, n\}, i > j + 1$ create an arc (b_j, b_i);

- for every $i \in \{1, 2, \ldots, n\}$ create an arc (a_i, b_i);
- for every $i, j \in \{1, 2, \ldots, n\}, i \neq j$ create an arc (b_j, a_i).

To see that T_n satisfies (i), observe that

$$\begin{pmatrix} \{b_1, b_2, \ldots, b_{n/2-1}\}, \\ \{a_{n/2+1}, a_{n/2+2}, \ldots, a_n\}, \\ \{a_1, a_2, \ldots, a_{n/2-1}\} \end{pmatrix}$$

is a $(n/2-1)$-triple. To prove that (ii) is satisfied as well, observe that there is a solution to Vertex Disjoint Paths problem containing two paths $(a_1, a_2, \ldots, a_n)$ and $(b_n, b_{n-1}, \ldots, b_1)$ which in total use all the vertices of T_n. Assume that there is some other solution and let k be the largest index such that the path connecting a_1 to a_n begins with prefix $(a_1, a_2, \ldots, a_k)$. As the solution is different from the aforementioned one, it follows that $k < n$. Therefore, the next vertex on the path has to be b_k, as this is the only unused outneighbor of a_k apart from a_{k+1}. But if the path from a_1 to a_n already uses $\{a_1, a_2, \ldots, a_k, b_k\}$, we see that there is no arc going from $\{a_{k+1}, a_{k+2}, \ldots, a_n, b_{k+1}, b_{k+2}, \ldots, b_n\}$ to $\{b_1, b_2, \ldots, b_{k-1}\}$, so we are already unable to construct a path from b_n to b_1. This is a contradiction.

The presented example suggests that a possible way of obtaining an FPT algorithm for Vertex Disjoint Paths problem requires another width parameter admitting more powerful obstacles.

Part III

In search for optimality

Chapter 8

Tight bounds for parameterized complexity of Cluster Editing

8.1 Introduction

Correlation clustering, also known as *clustering with qualitative information* or *cluster editing*, is the problem to cluster objects based only on the qualitative information concerning similarity between pairs of them. For every pair of objects we have a binary indication whether they are similar or not. The task is to find a partition of the objects into clusters minimizing the number of similarities between different clusters and non-similarities inside of clusters. The problem was introduced by Ben-Dor, Shamir, and Yakhini [28] motivated by problems from computational biology, and, independently, by Bansal, Blum, and Chawla [24], motivated by machine learning problems concerning document clustering according to similarities. The correlation version of clustering was studied intensively, including [5, 11, 17, 61, 62, 167, 299].

The graph-theoretic formulation of the problem is the following. A graph K is a *cluster graph* if every connected component of K is a complete graph. Let $G = (V, E)$ be a graph; then $F \subseteq V \times V$ is called a *cluster editing set* for G if $G \triangle F = (V, E \triangle F)$ is a cluster graph (recall that $E \triangle F$ denotes the symmetric difference between E and F). In the optimization version of the problem the task is to find a cluster editing set of minimum size. Constant factor approximation algorithms for this problem were obtained in [5, 24, 61]. On the negative side, the problem is known to be NP-complete [299] and, as was shown by Charikar, Guruswami, and Wirth [61], also APX-hard.

Giotis and Guruswami [167] initiated the study of clustering when the maximum number of clusters that we are allowed to use is stipulated to be a fixed constant p. As observed by them, this type of clustering is well-motivated in settings where the number of clusters might be an external constraint that has to be met. It appeared that p-clustering variants posed new and non-trivial challenges. In particular, in spite of the APX-hardness of the general case, Giotis and Guruswami [167] gave a PTAS for this version of the problem.

A cluster graph G is called a *p-cluster graph* if it has exactly p connected components or, equivalently, if it is a disjoint union of exactly p cliques. Similarly, a set F is a *p-cluster editing set* of G, if $G \triangle F$ is a p-cluster graph. In parameterized complexity, correlation clustering and its restriction to bounded number of clusters were studied under the names Cluster Editing and p-Cluster Editing, respectively.

Cluster Editing	
Input:	A graph $G = (V, E)$ and a nonnegative integer k.
Parameter:	k
Question:	Is there a cluster editing set for G of size at most k?

p-Cluster Editing	
Input:	A graph $G = (V, E)$ and nonnegative integers p and k.
Parameter:	p, k
Question:	Is there a p-cluster editing set for G of size at most k?

The parameterized version of Cluster Editing, and variants of it, were studied intensively [38, 39, 40, 41, 47, 98, 128, 168, 183, 184, 222, 271]. The problem is solvable in time $\mathcal{O}(1.62^k+n+m)$ [38] and it has a kernel with $2k$ vertices [57, 70]. Shamir et al. [299] showed that p-Cluster Editing is NP-complete for every fixed $p \geq 2$. A kernel with $(p+2)k+p$ vertices was given by Guo [182].

Our results. In this chapter, based on the results of [137], we study the impact of the interaction between p and k on the parameterized complexity of p-Cluster Editing. The main algorithmic result is the following.

Theorem 115. *p-Cluster Editing is solvable in time $\mathcal{O}(2^{\mathcal{O}(\sqrt{pk})}+m+n)$, where n is the number of vertices and m the number of edges of the input graph G.*

It is straightforward to modify the algorithm to work also in the following variants of the problem, where each edge and non-edge is assigned some edition cost: either (i) all costs are at least one and k is the bound on the maximum total cost of the solution, or (ii) we ask for a set of at most k edits of minimum cost. Let us also remark that, by Theorem 115, if $p = o(k)$ then p-Cluster Editing can be solved in $\mathcal{O}^*(2^{o(k)})$ time, and thus it belongs to complexity class SUBEPT defined by Flum and Grohe [132, Chapter 16]; see also Section 4.2 for a broader overview. Until very recently, the only natural problems known to be in the class SUBEPT were the problems with additional constraints on the input, like being a planar, H-minor-free, or tournament graph [10, 104]. However, recent algorithmic developments indicate that the structure of the class SUBEPT is much more interesting than expected. It appears that some parameterized problems related to edge modifications, like Minimum Fill-in or Split Edge Deletion, are also in SUBEPT [149, 165].

We would like to remark that p-Cluster Editing can be also solved in worse time complexity $\mathcal{O}((pk)^{\mathcal{O}(\sqrt{pk})} + m + n)$ using simple guessing arguments. One such algorithm is based on the following observation: Suppose that, for some integer r, we know at least $2r + 1$ vertices from each cluster. Then, if an unassigned vertex has at most r incident modifications, we know precisely to which cluster it belongs: it is adjacent to at least $r + 1$ vertices already assigned to its cluster and at most r assigned to any other cluster. On the other hand, there are at most $2k/r$ vertices with more than r incident modifications. Thus (i) guessing $2r + 1$ vertices from each cluster (or all of them, if there are less than $2r + 1$), and (ii) guessing all vertices with more than r incident modifications, together with their alignment to clusters, results in at most $n^{(2r+1)p}n^{2k/r}p^{2k/r}$ subcases. By pipelining it with the kernelization of Guo [182] and with simple reduction rules that ensure $p \leq 6k$ (see Section 8.3.1 for details), we obtain the claimed time complexity for $r \sim \sqrt{k/p}$.

An approach via *chromatic coding*, introduced by Alon et al. [10], also leads to an algorithm with running time $\mathcal{O}(2^{\mathcal{O}(p\sqrt{k}\log p)} + n + m)$. However, one needs to develop new concepts to construct an algorithm for p-Cluster Editing with complexity bound as promised in Theorem 115, and thus obtain a subexponential complexity for every sublinear p.

The crucial observation is that a p-cluster graph, for $p = \mathcal{O}(k)$, has $2^{\mathcal{O}(\sqrt{pk})}$ edge cuts of size at most k (i.e., k-cuts). As in a YES-instance to the p-Cluster Editing problem each k-cut is a $2k$-cut of a p-cluster graph, we infer a similar bound on the number of cuts if we are dealing with a YES-instance. This allows us to use dynamic programming over the set of k-cuts. Pipelining this approach with a kernelization algorithm for p-Cluster Editing proves Theorem 115.

Following the direction of pursuit of asymptotically tight bounds on the complexity of parameterized problems, described in Chapter 4, we complement Theorem 115 with two lower bounds. The first, main lower bound is based on the following technical Theorem 116, which shows that the exponential time dependence of the presented algorithm is asymptotically tight for any choice of parameters p and k, where $p = \mathcal{O}(k)$. As one can provide polynomial-time reduction rules that ensure that $p \leq 6k$ (see Section 8.3.1 for details), this provides a full and tight picture of the multivariate parameterized complexity of p-Cluster Editing: we have asymptotically matching upper and lower bounds on the whole interval between p being a constant and linear in k. To the best of our knowledge, this is the first fully multivariate and tight complexity analysis of a parameterized problem.

Theorem 116. *There is $\delta > 0$ and a polynomial-time algorithm that, given positive integers p and k and a 3-CNF-SAT formula Φ with n variables and m clauses, such that $k, n \geq p$ and $n, m \leq \sqrt{pk}$, computes a graph G and integer k', such that $k' \leq \delta k$, $|V(G)| \leq \delta\sqrt{pk}$, and*

- *if Φ is satisfiable then there is a $6p$-cluster graph G_0 with $V(G) = V(G_0)$ and $|E(G) \triangle E(G_0)| \leq k'$;*
- *if there exists a p'-cluster graph G_0 with $p' \leq 6p$, $V(G) = V(G_0)$ and $|E(G) \triangle E(G_0)| \leq k'$, then Φ is satisfiable.*

As the statement of Theorem 116 may look technical, we gather its two main consequences in Corollaries 117 and 118. We state both corollaries in terms of an easier $p_{\leq}$-Cluster Editing problem, where the number of clusters has to be at most p instead of precisely equal to p. Clearly, this version can be solved by an algorithm for p-Cluster Editing with an additional p overhead in time complexity by trying all possible $p' \leq p$, so the lower bound holds also for harder p-Cluster Editing; however, we are not aware of any reduction in the opposite direction. In both corollaries we use the fact that existence of a subexponential, in both the number of variables and clauses, algorithm for verifying satisfiability of 3-CNF-SAT formulas would violate ETH (Corollary 5).

Corollary 117. *Unless ETH fails, for every $0 \leq \sigma \leq 1$ there is $p(k) \in \Theta(k^\sigma)$ such that $p_{\leq}$-Cluster Editing restricted to instances where $p = p(k)$ is not solvable in time $\mathcal{O}^*(2^{o(\sqrt{pk})})$.*

Proof. Assume we are given a 3-CNF-SAT formula Φ with n variables and m clauses. If $n < m$, then $\lceil (m-n)/2 \rceil$ times perform the following operation: add three new variables x, y and z, and clause $(x \vee y \vee z)$. In this way we preserve the satisfiability of Φ, increase the size of Φ at most by a constant multiplicative factor, and ensure that $n \geq m$.

Take now $k = \lceil n^{\frac{2}{1+\sigma}} \rceil$, $p = \lceil n^{\frac{2\sigma}{1+\sigma}} \rceil$. As $n \geq m$ and $0 \leq \sigma \leq 1$, we have $k, n \geq p$ and $n, m \leq \sqrt{pk}$ but $n + m = \Omega(\sqrt{pk})$. Invoke Theorem 116 for the formula Φ and parameters p and k, obtaining a graph G and a parameter k'. Note that $6p \in \Theta(k^\sigma)$. Apply the assumed algorithm for the

$p_{\leq}$-Cluster Editing problem to the instance $(G, 6p, k')$. In this way we resolve the satisfiability of Φ in time $\mathcal{O}^*(2^{o(\sqrt{pk})}) = 2^{o(n+m)}$, which contradicts ETH by Corollary 5. □

Corollary 118. *Unless ETH fails, for every constant $p \geq 6$ there is no algorithm solving $p_{\leq}$-Cluster Editing in time $\mathcal{O}^*(2^{o(\sqrt{k})})$ or $2^{o(|V(G)|)}$.*

Proof. We prove the corollary for $p = 6$; the claim for larger values of p can be proved easily by taking the graph obtained in the reduction and introducing additional $p - 6$ cliques of its size.

Assume we are given a 3-CNF-SAT formula Φ with n variables and m clauses. Take $k = \max(n, m)^2$, invoke Theorem 116 for the formula Φ and parameters 1 and k, obtaining a graph G and a parameter $k' = \mathcal{O}(k)$. Note that $|V(G)| = \mathcal{O}(\sqrt{k})$. Apply the assumed algorithm for the $p_{\leq}$-Cluster Editing problem to the instance $(G, 6, k')$. In this way we resolve the satisfiability of Φ in time $\mathcal{O}^*(2^{o(\sqrt{k})}) = 2^{o(n+m)}$, contradicting ETH. □

Note that Theorem 116 and Corollary 117 do not rule out possibility that the general Cluster Editing is solvable in subexponential time. Our second, complementary lower bound shows that when the number of clusters is not constrained, then the problem cannot be solved in subexponential time unless ETH fails. This disproves the conjecture of Cao and Chen [57]. We note that Theorem 119 was independently obtained by Komusiewicz and Uhlmann [222]; however, we choose to include it for the sake of completeness of our study.

Theorem 119. *Unless ETH fails,* Cluster Editing *cannot be solved in time $\mathcal{O}^*(2^{o(k)})$.*

Clearly, by Theorem 115 the reduction of Theorem 119 must produce an instance where the number of clusters in any solution, if there exists any, is $\Omega(k)$. Therefore, intuitively the hard instances of Cluster Editing are those where every cluster needs just a constant number of adjacent editions to be extracted.

Organization of the chapter. In Section 8.2 we recall some results that will be of use later. Section 8.3 contains description of the subexponential algorithm for p-Cluster Editing, i.e., the proof of Theorem 115. Section 8.4 is devoted to the multivariate lower bound, i.e., the proof of Theorem 116, while in Section 8.5 we give the lower bound for the general Cluster Editing problem, i.e., the proof of Theorem 119. In Section 8.6 we gather some concluding remarks and propositions for further work.

8.2 Preliminaries

In this chapter we will use notation uv both to denote edges and non-edges, that is, unordered pairs of different vertices that are not contained in the edge set of the given graph. To simplify the notation, we will also often treat sets of ordered pairs as sets of unordered pairs. Thus, for instance, for vertex sets X, Y by $X \times Y$ we denote the set of all xy such that $x \in X$ and $y \in Y$.

In our algorithm we need the following result of Guo [182].

Theorem 120 ([182])**.** *p-Cluster Editing admits a kernel with at most $(p+2)k + p$ vertices. More precisely, there exists an algorithm that, given an instance (G, p, k) of p-Cluster Editing such that $|V(G)| = n$ and $|E(G)| = m$, runs in time $\mathcal{O}(n + m)$ and outputs an equivalent instance (G', p', k') of p-Cluster Editing such that $k' \leq k$, $p' \leq p$, and $|V(G')| \leq (p'+2)k' + p'$.*

We remark that Guo [182] does not mention explicitely that the algorithm can only decrease values of p and k, but this can be easily checked by examining the single reduction rule presented in [182].

The following lemma is used in both our lower bounds. Its proof is almost identical to the proof of Lemma 1 in [182], and we provide it here for reader's convenience.

Lemma 121. *Let $G = (V, E)$ be an undirected graph and $X \subseteq V$ be a set of vertices such that $G[X]$ is a clique and each vertex in X has the same set of neighbors outside X (i.e., $N_G[v] = N_G[X]$ for each $v \in X$). Let $F \subseteq V \times V$ be a set such that $G \triangle F$ is a cluster graph where the vertices of X are in at least two different clusters. Then there exists $F' \subseteq V \times V$ such that: (i) $|F'| < |F|$, (ii) $G \triangle F'$ is a cluster graph with no larger number of clusters than $G \triangle F$, (iii) in $G \triangle F'$ the clique $G[X]$ is contained in one cluster.*

Proof. For a vertex $v \in X$, let $F(v) = \{u \notin X : vu \in F\}$. Note that, since $N_G[v] = N_G[X]$ for all $v \in X$, we have $F(v) = F(w)$ if v and w belong to the same cluster in $G \triangle F$.

Let Z be the vertex set of a cluster in $G \triangle F$ such that there exists $v \in Z \cap X$ with smallest $|F(v)|$. Construct F' as follows: take F, and for each $w \in X$ replace all elements of F incident with w with $\{uw : u \in F(v)\}$. In other words, we modify F by moving all vertices of $X \setminus Z$ to the cluster Z. Clearly, $G \triangle F'$ is a cluster graph, X is contained in one cluster in $G \triangle F'$ and $G \triangle F'$ contains no more clusters than $G \triangle F$. To finish the proof, we need to show that $|F'| < |F|$. The sets F and F' contain the same set of elements not incident with X. As $|F(v)|$ was minimum possible, for each $w \in X$ we have $|F(w)| \geq |F'(w)|$. As X was split between at least two connected components of $G \triangle F$, F contains at least one edge of $G[X]$, whereas F' does not. We infer that $|F'| < |F|$ and the lemma is proven. □

8.3 A subexponential algorithm

In this section we prove Theorem 115, i.e., we show an algorithm for p-Cluster Editing running in time $\mathcal{O}(2^{\mathcal{O}(\sqrt{pk})} + n + m)$.

8.3.1 Reduction for large p

The first step of our algorithm is an application of the kernelization algorithm by Guo [182] (Theorem 120) followed by some additional preprocessing rules that ensure that $p \leq 6k$. These additional rules are encapsulated in the following lemma; the rest of this section is devoted to its proof.

Lemma 122. *There exists a polynomial time algorithm that, given an instance (G, p, k) of p-Cluster Editing, outputs an equivalent instance (G', p', k), where G' is an induced subgraph of G and $p' \leq 6k$.*

Before we proceed to formal argumentation, let us provide some intuition. The key idea behind Lemma 122 is an observation that if $p > 2k$, then at least $p - 2k$ clusters in the final cluster graph cannot be touched by the solution, hence they must have been present as isolated cliques already in the beginning. Hence, if $p > 6k$ then we have to already see $p - 2k > 4k$ isolated cliques; otherwise, we may safely provide a negative answer. Although these cliques may be still merged (to decrease the number of clusters) or split (to increase the number of clusters), we can apply greedy arguments to identify a clique that may be safely assumed to be untouched by the solution. Hence

we can remove it from the graph and decrement p by one. Although the greedy arguments seem very intuitive, their formal proofs turn out to be somewhat technical.

We now proceed to a formal proof of Lemma 122. Let us fix some optimal solution F, i.e., a subset of $V \times V$ of minimum cardinality such that $G \triangle F$ is a p-cluster graph.

Consider the case when $p > 6k$. Observe that only $2k$ out of p resulting clusters in $G \triangle F$ can be adjacent to any pair from the set F. Hence at least $p - 2k$ clusters must be already present in the graph G as connected components being cliques. Therefore, if G contains less than $p - 2k$ connected components that are cliques, then (G, p, k) is a NO-instance.

Rule 1. *If G contains less than $p - 2k$ connected components that are cliques, answer NO.*

As $p > 6k$, if Rule 1 was not triggered then we have more than $4k$ connected components that are cliques. The aim is now to apply greedy arguments to identify a component that can be safely assumed to be untouched. As a first step, consider a situation when G contains more than $2k$ isolated vertices. Then at least one of these vertices is not incident to an element of F, thus we may delete one isolated vertex and decrease p by one.

Rule 2. *If G contains $2k+1$ isolated vertices, pick one of them, say v, and delete it from G. The new instance is $(G \setminus v, p-1, k)$.*

Safeness of this rule is straightforward. We are left with the case where G contains more than $2k$ connected components that are cliques, but not isolated vertices. At least one of these cliques is untouched by F. Note that even though the number of cliques is large, some of them may be merged with other clusters (to decrease the number of connected components), or split into more clusters (to increase the number of connected components), and we have no a priori knowledge about which clique will be left untouched. We argue that in both cases, we can greedily merge or split the smallest possible clusters. Thus, without loss of generality, we can assume that the largest connected component of G that is a clique is left untouched in $G \triangle F$. We reduce the input instance (G, p, k) by deleting this cluster and decreasing p by one.

Rule 3. *If G contains $2k+1$ isolated cliques that are not isolated vertices, pick a clique C of largest size and delete it from G. The new instance is $(G \setminus C, p-1, k)$.*

We formally verify safeness of Rule 3 by proving the following lemma. Without loss of generality, we may assume that the solution F, among all the solutions of minimum cardinality, has minimum possible number of editions incident to vertices of connected components of G that are cliques of largest size.

Lemma 123. *Let $D_1, D_2, \ldots, D_\ell$ be connected components of G that are cliques, but not isolated vertices. Assume that $\ell \geq 2k+1$. Then there exists a component D_i that has the largest size among $D_1, D_2, \ldots, D_\ell$ and none of the pairs from F is incident to any vertex of D_i.*

Proof. Let $C_1, C_2, \ldots, C_p$ be clusters of $G \triangle F$. We say that cluster C_i *contains* component D_j if $V(D_j) \subseteq V(C_i)$, and component D_j *contains* cluster C_i if $V(C_i) \subseteq V(D_j)$. Moreover, we say that these containments are *strict* if $V(D_j) \subsetneq V(C_i)$ or $V(C_i) \subsetneq V(D_j)$, respectively.

Claim 1. *For every cluster C_i and component D_j, either $V(C_i)$ and $V(D_j)$ are disjoint, or C_i contains D_j, or D_j contains C_i.*

Proof. We need to argue that the situation when sets $V(C_i) \cap V(D_j)$, $V(C_i) \setminus V(D_j)$, $V(D_j) \setminus V(C_i)$ are simultaneously nonempty is impossible. Assume otherwise, and without loss of generality assume further that $|V(C_i) \setminus V(D_j)|$ is largest possible among pairs (C_i, D_j) satisfying the stated condition. As $V(D_j) \setminus V(C_i) \neq \emptyset$, take some $C_{i'} \neq C_i$ such that $V(C_{i'}) \cap V(D_j) \neq \emptyset$. By the choice of C_i we have that $|V(C_i) \setminus V(D_j)| \geq |V(C_{i'}) \setminus V(D_j)|$ (note that $V(C_{i'}) \setminus V(D_j)$ is possibly empty). Consider a new cluster graph H obtained from $G \triangle F$ by moving $V(C_i) \cap V(D_j)$ from the cluster C_i to the cluster $C_{i'}$. Clearly, H still has p clusters as $V(C_i) \setminus V(D_j)$ is nonempty. Moreover, the edition set F' that needs to be modified in order to obtain H from G, differs from F as follows: it additionally contains $(V(C_i) \cap V(D_j)) \times (V(C_{i'}) \setminus V(D_j))$, but does not contain $(V(C_i) \cap V(D_j)) \times (V(C_i) \setminus V(D_j))$ nor $(V(C_i) \cap V(D_j)) \times (V(C_{i'}) \cap V(D_j))$. As $|V(C_i) \setminus V(D_j)| \geq |V(C_{i'}) \setminus V(D_j)|$, we have that

$$|(V(C_i) \cap V(D_j)) \times (V(C_{i'}) \setminus V(D_j))| \leq |(V(C_i) \cap V(D_j)) \times (V(C_i) \setminus V(D_j))|$$

and

$$|(V(C_i) \cap V(D_j)) \times (V(C_{i'}) \cap V(D_j))| > 0.$$

Hence $|F'| < |F|$, which is a contradiction with minimality of F. This settles Claim 1. ⌟

We say that a component D_j is *embedded* if some cluster C_i strictly contains it. Moreover, we say that a component D_j is *broken* if it strictly contains more than one cluster; Claim 1 implies that then $V(D_j)$ is the union of vertex sets of the clusters it strictly contains. Component D_j is *untouched* if none of the pairs from F is incident to a vertex from D_j. Claim 1 proves that every component D_j is either embedded, broken or untouched.

Claim 2. *It is impossible that some component D_j is broken and some other $D_{j'}$ is embedded.*

Proof. Aiming towards a contradiction assume that some component D_j is broken and some other $D_{j'}$ is embedded. Let C_{i_1}, C_{i_2} be any two clusters contained in D_j and let $C_{i'}$ be the cluster that strictly contains $D_{j'}$. Consider a new cluster graph H obtained from $G \triangle F$ by merging clusters C_{i_1},C_{i_2} and splitting cluster $C_{i'}$ into clusters on vertex sets $V(C_{i'}) \setminus V(D_{j'})$ and $V(D_{j'})$. As $V(C_{i'}) \setminus V(D_{j'}) \neq \emptyset$, H is still a p-cluster graph. Moreover, the edition set F' that need to be modified in order to obtain H from G, differs from F by not containing $V(C_{i_1}) \times V(C_{i_2})$ and not containing $(V(C_{i'}) \setminus V(D_{j'})) \times V(D_{j'})$. Both of this sets are nonempty and hence $|F'| < |F|$, which is a contradiction with minimality of F. This settles Claim 2. ⌟

Claim 2 implies that either none of the components is broken, or none is embedded. We firstly prove that in the first case the lemma holds. Note that as $\ell > 2k$, at least one component D_j is untouched.

Claim 3. *If none of the components $D_1, D_2, \ldots, D_\ell$ is broken, then there is an untouched component D_j with the largest number of vertices among $D_1, D_2, \ldots, D_\ell$.*

Proof. Aiming towards a contradiction assume that all the components with largest numbers of vertices are not untouched, hence they are embedded. Take any such component D_j and let $D_{j'}$ be any untouched component; by the assumption we infer that $|V(D_j)| > |V(D_{j'})|$. Let C_i be the cluster that strictly contains D_j and let $C_{i'}$ be the cluster corresponding to the (untouched) component $D_{j'}$. Consider a cluster graph H obtained from $G \triangle F$ by exchanging sets $V(D_j)$ and $V(D_{j'})$ between clusters C_i and $C_{i'}$. Observe that the edition set F' that needs to be modified in order to obtain H from G, differs from F by not containing $(V(C_i) \setminus V(D_j)) \times V(D_j)$ but containing

$(V(C_i) \setminus V(D_j)) \times V(D_{j'})$. However, $|V(D_j)| > |V(D_{j'})|$ and $|V(C_i) \setminus V(D_j)| > 0$, so $|F'| < |F|$. This contradicts minimality of F and settles Claim 3. ⌟

We are left with the case when all the clusters are broken or untouched.

Claim 4. *If none of the components $D_1, D_2, \ldots, D_\ell$ is embedded, then there is an untouched component D_j with the largest number of vertices among $D_1, D_2, \ldots, D_\ell$.*

Proof. Aiming towards a contradiction assume that all the components with largest numbers of vertices are not untouched, hence they are broken. Take any such component D_j and let $D_{j'}$ be any untouched component; by the assumption we infer that $|V(D_j)| > |V(D_{j'})|$. Assume that D_j is broken into $q+1$ clusters ($q \geq 1$) of sizes $a_1, a_2, \ldots, a_{q+1}$, where $\sum_{i=1}^{q+1} a_i = |V(D_j)|$. The number of editions needed inside component D_j is hence equal to

$$\begin{aligned}\binom{|V(D_j)|}{2} - \sum_{i=1}^{q+1} \binom{a_i}{2} &\geq \binom{|V(D_j)|}{2} - \binom{|V(D_j)|-q}{2} - q\binom{1}{2} \\ &= \binom{|V(D_j)|}{2} - \binom{|V(D_j)|-q}{2}.\end{aligned}$$

The inequality follows from convexity of function $t \to \binom{t}{2}$. We now consider two cases.

Assume first that $|V(D_{j'})| > q$. Let us change the edition set F into F' by altering editions inside components D_j and $D_{j'}$ as follows: instead of breaking D_j into $q+1$ components and leaving $D_{j'}$ untouched, leave D_j untouched and break $D_{j'}$ into $q+1$ components by creating q singleton clusters and one cluster of size $|V(D_{j'})| - q$. Similar calculations to the ones presented in the paragraph above show that the edition cost inside components D_j and $D_{j'}$ is equal to $\binom{|V(D_{j'})|}{2} - \binom{|V(D_{j'})|-q}{2} < \binom{|V(D_j)|}{2} - \binom{|V(D_j)|-q}{2}$. Hence, we can obtain the same number of clusters with a strictly smaller edition set, a contradiction with minimality of F.

Assume now that $|V(D_{j'})| \leq q$. Let us change the edition set F into F' by altering editions inside components D_j and $D_{j'}$ as follows: instead of breaking D_j into $q+1$ components and leaving $D_{j'}$ untouched, we break $D_{j'}$ totally into $|V(D_{j'})|$ singleton clusters and break D_j into $q-|V(D_{j'})|+1$ singleton clusters and one of size $|V(D_j)|-q+|V(D_{j'})|-1$. Clearly, we get the same number of clusters in this manner. Similar calculations as before show that the number of new editions needed inside clusters D_j and $D_{j'}$ is equal to $\binom{|V(D_{j'})|}{2} + \binom{|V(D_j)|}{2} - \binom{|V(D_j)|-q+|V(D_{j'})|-1}{2}$; it may be readily checked that this value is always not larger than $\binom{|V(D_j)|}{2} - \binom{|V(D_j)|-q}{2}$ for $|V(D_{j'})| \geq 1$ and $|V(D_j)| \geq q+1$. Recall now that components $D_1, D_2, \ldots, D_\ell$ are not independent vertices, so $|V(D_{j'})| \geq 2$ and F' is obtained by removing from F some editions that were adjacent to the vertex set of D_j, and inserting at most the same number of editions, at least one of which being adjacent only to vertices of $D_{j'}$. Hence, we can obtain the same number of clusters with a not larger edition set and with a smaller number of editions incident to components of G that are cliques of largest size. This contradicts the choice of F. We have obtained a contradiction in both cases, so Claim 4 follows. ⌟

Lemma 123 follows directly from Claims 1, 2, 3, and 4. □

Clearly, an instance on which none of the Rules 1–3 may be triggered has $p \leq 6k$. This proves Lemma 122.

8.3.2 Bounds on binomial coefficients

In the running time analysis we need some combinatorial bounds on binomial coefficients. More precisely, we use the following inequality.

Lemma 124. *If a, b are nonnegative integers, then $\binom{a+b}{a} \leq 2^{2\sqrt{ab}}$.*

We start with the following simple observation.

Lemma 125. *If a, b are positive integers, then $\binom{a+b}{a} \leq \frac{(a+b)^{a+b}}{a^a b^b}$.*

Proof. In the proof we use a folklore fact that the sequence $a_n = (1+1/n)^n$ is increasing. This implies that $\left(1+\frac{1}{b}\right)^b \leq \left(1+\frac{1}{a+b}\right)^{a+b}$, equivalently $\frac{(a+b)^{a+b}}{b^b} \leq \frac{(a+b+1)^{a+b}}{(b+1)^b}$.

Let us fix a; we prove the claim via induction with respect to b. For $b = 1$ the claim is equivalent to $a^a \leq (a+1)^a$ and therefore trivial. In order to check the induction step, notice that

$$\begin{aligned}\binom{a+b+1}{a} &= \frac{a+b+1}{b+1} \cdot \binom{a+b}{a} \leq \frac{a+b+1}{b+1} \cdot \frac{(a+b)^{a+b}}{a^a b^b} \\ &\leq \frac{a+b+1}{b+1} \cdot \frac{(a+b+1)^{a+b}}{a^a (b+1)^b} = \frac{(a+b+1)^{a+b+1}}{a^a (b+1)^{b+1}}.\end{aligned}$$

□

We proceed to the proof of Lemma 124.

Proof of Lemma 124. Firstly, observe that the claim is trivial for $a = 0$ or $b = 0$; hence, we can assume that $a, b > 0$. Moreover, without losing generality assume that $a \leq b$. Let us denote $\sqrt{ab} = \ell$ and $\frac{a}{b} = t$, then $0 < t \leq 1$. By Lemma 125 we have that

$$\begin{aligned}\binom{a+b}{a} &\leq \frac{(a+b)^{a+b}}{a^a b^b} = \left(1+\frac{b}{a}\right)^a \cdot \left(1+\frac{a}{b}\right)^b \\ &= \left[\left(1+\frac{1}{t}\right)^{\frac{\sqrt{t}}{1}} \cdot \left(1+\frac{t}{1}\right)^{\frac{1}{\sqrt{t}}}\right]^\ell.\end{aligned}$$

Let us denote $g(x) = x \ln\left(1+x^{-2}\right) + x^{-1} \ln\left(1+x^2\right)$. As $\binom{a+b}{a} \leq e^{\ell \cdot g\left(\sqrt{t}\right)}$, it suffices to prove that $g(x) \leq 2\ln 2$ for all $0 < x \leq 1$. Observe that

$$\begin{aligned}g'(x) &= \ln\left(1+x^{-2}\right) - x \cdot 2x^{-3} \cdot \frac{1}{1+x^{-2}} - x^{-2} \ln\left(1+x^2\right) + x^{-1} \cdot 2x \cdot \frac{1}{1+x^2} \\ &= \ln\left(1+x^{-2}\right) - \frac{2}{1+x^2} - x^{-2} \ln\left(1+x^2\right) + \frac{2}{1+x^2} \\ &= \ln\left(1+x^{-2}\right) - x^{-2} \ln\left(1+x^2\right).\end{aligned}$$

Let us now introduce $h : (0,1] \to \mathbb{R}$, $h(y) = g'(\sqrt{y}) = \ln\left(1+y^{-1}\right) - y^{-1} \ln(1+y)$. Then,

$$\begin{aligned}h'(y) &= -y^{-2} \cdot \frac{1}{1+y^{-1}} + y^{-2} \ln(1+y) - y^{-1} \cdot \frac{1}{1+y} \\ &= y^{-2} \ln(1+y) - \frac{2}{y+y^2}.\end{aligned}$$

We claim that $h'(y) \leq 0$ for all $y \leq 1$. Indeed, from the inequality $\ln(1+y) \leq y$ we infer that

$$y^{-2} \ln(1+y) \quad \leq \quad y^{-1} = \frac{2}{y+y} \leq \frac{2}{y+y^2}.$$

Therefore, $h'(y) \leq 0$ for $y \in (0,1]$, so $h(y)$ is non-increasing on this interval. As $h(1) = 0$, this implies that $h(y) \geq 0$ for $y \in (0,1]$, so also $g'(x) \geq 0$ for $x \in (0,1]$. This means that $g(x)$ is non-decreasing on the interval $(0,1]$, so $g(x) \leq g(1) = 2\ln 2$. □

8.3.3 Small cuts

We now proceed to the algorithm itself. Let us introduce the key notion, whose directed analogue was considered in Section 6.3.

Definition 126 (see also Definition 73)**.** *Let $G = (V,E)$ be an undirected graph. A partition (V_1, V_2) of V is called a* k-cut of G *if $|E(V_1, V_2)| \leq k$.*

Lemma 127 (see also Lemma 74)**.** *k-cuts of a graph G can be enumerated with polynomial-time delay.*

Proof. We follow a standard branching. We order the vertices arbitrarily, start with empty V_1, V_2 and for each consecutive vertex v we branch into two subcases: we put v either into V_1 or into V_2. Once the alignment of all vertices is decided, we output the cut. However, each time we put a vertex in one of the sets, we run a polynomial-time max-flow algorithm to check whether the minimum edge cut between V_1 and V_2 constructed so far is at most k. If not, then we terminate this branch as it certainly cannot result in any solutions found. Thus, we always pursue a branch that results in at least one feasible solution, and finding the next solution occurs within a polynomial number of steps. □

Intuitively, k-cuts of the graph G form the search space of the algorithm. Therefore, we would like to bound their number. We do this by firstly bounding the number of cuts of a cluster graph, and then using the fact that a YES-instance is not very far from some cluster graph.

Lemma 128. *Let K be a cluster graph containing at most p clusters, where $p \leq 6k$. Then the number of k-cuts of K is at most $2^{8\sqrt{pk}}$.*

Proof. By slightly abusing the notation, assume that K has exactly p clusters, some of which may be empty. Let $C_1, C_2, \ldots, C_p$ be these clusters and $c_1, c_2, \ldots, c_p$ be their sizes, respectively. We firstly establish a bound on the number of cuts (V_1, V_2) such that the cluster C_i contains x_i vertices from V_1 and y_i from V_2. Then we discuss how to bound the number of ways of selecting pairs x_i, y_i summing up to c_i for which the number of k-cuts is positive. Multiplying the obtained two bounds gives us the claim.

Having fixed the numbers x_i, y_i, the number of ways in which the cluster C_i can be partitioned is equal to $\binom{x_i+y_i}{x_i}$. Note that $\binom{x_i+y_i}{x_i} \leq 2^{2\sqrt{x_i y_i}}$ by Lemma 124. Observe that there are $x_i y_i$ edges between V_1 and V_2 inside the cluster C_i, so if (V_1, V_2) is a k-cut, then it follows that $\sum_{i=1}^{p} x_i y_i \leq k$. By applying the classical Cauchy-Schwarz inequality we infer that $\sum_{i=1}^{p} \sqrt{x_i y_i} \leq \sqrt{p} \cdot \sqrt{\sum_{i=1}^{p} x_i y_i} \leq \sqrt{pk}$. Therefore, the number of considered cuts is bounded by

$$\prod_{i=1}^{p} \binom{x_i + y_i}{x_i} \leq 2^{2\sum_{i=1}^{p} \sqrt{x_i y_i}} \leq 2^{2\sqrt{pk}}.$$

Moreover, observe that $\min(x_i, y_i) \leq \sqrt{x_i y_i}$; hence, $\sum_{i=1}^{p} \min(x_i, y_i) \leq \sqrt{pk}$. Thus, the choice of x_i, y_i can be modeled by first choosing for each i, whether $\min(x_i, y_i)$ is equal to x_i or to y_i, and then expressing $\lfloor \sqrt{pk} \rfloor$ as the sum of $p+1$ nonnegative numbers: $\min(x_i, y_i)$ for $1 \leq i \leq p$ and the rest, $\lfloor \sqrt{pk} \rfloor - \sum_{i=1}^{p} \min(x_i, y_i)$. The number of choices in the first step is equal to $2^p \leq 2^{\sqrt{6pk}}$, and in the second is equal to $\binom{\lfloor \sqrt{pk} \rfloor + p}{p} \leq 2^{\sqrt{pk}+\sqrt{6pk}}$. Therefore, the number of possible choices of x_i, y_i is bounded by $2^{(1+2\sqrt{6})\sqrt{pk}} \leq 2^{6\sqrt{pk}}$. Hence, the total number of k-cuts is bounded by $2^{6\sqrt{pk}} \cdot 2^{2\sqrt{pk}} = 2^{8\sqrt{pk}}$, as claimed. □

Armed with Lemma 128, we are ready to prove a bound on the number of k-cuts.

Lemma 129. *If (G, p, k) is a YES-instance of p-Cluster Editing with $p \leq 6k$, then the number of k-cuts of G is bounded by $2^{8\sqrt{2pk}}$.*

Proof. Let K be a cluster graph with at most p clusters such that $\mathcal{H}(G, K) \leq k$. Observe that every k-cut of G is also a $2k$-cut of K, as K differs from G by at most k edge modifications. The claim follows from Lemma 128. □

8.3.4 The algorithm

We are finally ready to present the full algorithm, that is, to prove Theorem 115.

Proof of Theorem 115. Let $(G = (V, E), p, k)$ be the given p-Cluster Editing instance. By making use of Theorem 120 we can assume that G has at most $(p+2)k + p$ vertices, thus all the factors polynomial in the size of G can be henceforth hidden within the $2^{\mathcal{O}(\sqrt{pk})}$ factor. Application of Theorem 120 gives the additional $\mathcal{O}(n+m)$ summand to the complexity. By further usage of Lemma 122 we can also assume that $p \leq 6k$. Note that application of Lemmas 122 and 120 can only decrease the original parameters p, k. Note also that application of Lemma 122 can spoil the bound $|V| \leq (p+2)k + p$ as p can decrease; however the number of vertices of the graph is still bounded in terms of original p and k.

We now enumerate k-cuts of G with polynomial time delay. If during enumeration we exceed the bound $2^{8\sqrt{2pk}}$ given by Lemma 129, we know that we can safely answer NO, so we immediately terminate the computation and give a negative answer. Therefore, we can assume that we have computed the set $\mathcal{N}$ of all k-cuts of G and $|\mathcal{N}| \leq 2^{8\sqrt{2pk}}$.

Assume that (G, p, k) is a YES-instance and let K be a cluster graph with at most p clusters such that $\mathcal{H}(G, K) \leq k$. Again, let $C_1, C_2, \ldots, C_p$ be the clusters of K. Observe that for every $j \in \{0, 1, 2, \ldots, p\}$, the partition $\left(\bigcup_{i=1}^{j} V(C_i), \bigcup_{i=j+1}^{p} V(C_i)\right)$ has to be the k-cut with respect to G, as otherwise there would be more than k edges that need to be deleted from G in order to obtain K. This observation enables us to use a dynamic programming approach on the set of cuts.

We construct a directed graph D, whose vertex set is equal to $\mathcal{N} \times \{0, 1, 2, \ldots, p\} \times \{0, 1, 2, \ldots, k\}$; note that $|V(D)| = 2^{\mathcal{O}(\sqrt{pk})}$. We create arcs going from $((V_1, V_2), j, \ell)$ to $((V_1', V_2'), j+1, \ell')$, where $V_1 \subsetneq V_1'$ (hence $V_2 \supsetneq V_2'$), $j \in \{0, 1, 2, \ldots, p-1\}$ and $\ell' = \ell + |E(V_1, V_1' \setminus V_1)| + |\overline{E}(V_1' \setminus V_1, V_1' \setminus V_1)|$ ($(V, \overline{E})$ is the complement of the graph G). The arcs can be constructed in $2^{\mathcal{O}(\sqrt{pk})}$ time by checking for all the pairs of vertices whether they should be connected. We claim that the answer to the instance (G, p, k) is equivalent to reachability of any of the vertices of form $((V, \emptyset), p, \ell)$ from the vertex $((\emptyset, V), 0, 0)$.

In one direction, if there is a path from $((\emptyset, V), 0, 0)$ to $((V, \emptyset), p, \ell)$ for some $\ell \leq k$, then the consecutive sets $V_1' \setminus V_1$ along the path form clusters C_i of a cluster graph K, whose editing distance

to G is accumulated on the last coordinate, and is thus bounded by k. In the second direction, if there is a cluster graph K with clusters $C_1, C_2, \ldots, C_p$ within editing distance at most k from G, then vertices of form

$$\left(\left(\bigcup_{i=1}^{j} V(C_i), \bigcup_{i=j+1}^{p} V(C_i)\right), j, \mathcal{H}\left(G\left[\bigcup_{i=1}^{j} V(C_i)\right], K\left[\bigcup_{i=1}^{j} V(C_i)\right]\right)\right)$$

constitute a path from $((\emptyset, V), 0, 0)$ to $((V, \emptyset), p, \mathcal{H}(G, K))$. Note that all these triples are indeed vertices of the graph D, since all the partitions of the form $\left(\bigcup_{i=1}^{j} V(C_i), \bigcup_{i=j+1}^{p} V(C_i)\right)$ are k-cuts of G.

Reachability in a directed graph can be tested in linear time with respect to the number of vertices and arcs. We can now apply this algorithm to the digraph D and conclude solving the p-Cluster Editing instance in total $\mathcal{O}(2^{\mathcal{O}(\sqrt{pk})} + n + m)$ time. □

8.4 Multivariate lower bound

This section contains the proof of Theorem 116. The proof consists of four parts. In Section 8.4.1 we preprocess the input formula Φ to make it more regular. Section 8.4.2 contains the details of the construction of the graph G. In Section 8.4.3 we show how to translate a satisfying assignment of Φ into a $6p$-cluster graph G_0 close to G and we provide a reverse implication in Section 8.4.4. The constants hidden in the $\mathcal{O}$-notation correspond to the factor δ in the statement of Theorem 116.

8.4.1 Preprocessing of the formula

We start with a step that regularizes the input formula Φ, while increasing its size only by a constant multiplicative factor. The purpose of this step is to ensure that, when we translate a satisfying assignment of Φ into a cluster graph G_0 in the completeness step, the clusters are of the same size, and therefore contain the minimum possible number of edges. This property is used in the argumentation of the soundness step.

Lemma 130. *There exists a polynomial-time algorithm that, given a 3-CNF formula Φ with n variables and m clauses and an integer p, $p \leq n$, constructs a 3-CNF formula Φ' with n' variables and m' clauses together with a partition of the variable set $\mathrm{Vars}(\Phi')$ into p parts Vars^r, $1 \leq r \leq p$, such that following properties hold:*

(a) Φ' is satisfiable iff Φ is;

(b) in Φ' every clause contains exactly three literals corresponding to different variables;

(c) in Φ' every variable appears exactly three times positively and exactly three times negatively;

(d) n' is divisible by p and, for each $1 \leq r \leq p$, $|\mathrm{Vars}^r| = n'/p$ (i.e., the variables are split evenly between the parts Vars^r);

(e) if Φ' is satisfiable, then there exists a satisfying assignment of $\mathrm{Vars}(\Phi')$ with the property that in each part Vars^r the numbers of variables set to true and to false are equal.

(f) $n' + m' = \mathcal{O}(n + m)$.

Proof. We modify Φ while preserving satisfiability, consecutively ensuring that properties (b), (c), (d), and (e) are satisfied. Satisfaction of (f) will follow directly from the constructions used.

First, delete every clause that contains two different literals corresponding to the same variable, as they are always satisfied. Remove copies of the same literals inside clauses. Until all the clauses have at least two literals, remove every clause containing one literal, set the value of this literal so that the clause is satisfied and propagate this knowledge to the other clauses. At the end, create a new variable p and for every clause C that has two literals replace it with two clauses $C \vee p$ and $C \vee \neg p$. All these operations preserve satisfiability and at the end all the clauses consist of exactly three different literals.

Second, duplicate each clause so that every variable appears an even number of times. Introduce two new variables q, r. Take any variable x, assume that x appears positively k^+ times and negatively k^- times. If $k^+ < k^-$, introduce clauses $(x \vee q \vee r)$ and $(x \vee \neg q \vee \neg r)$, each $\frac{k^- - k^+}{2}$ times, otherwise introduce clauses $(\neg x \vee q \vee r)$ and $(\neg x \vee \neg q \vee \neg r)$, each $\frac{k^+ - k^-}{2}$ times. These operations preserve satisfiability (as the new clauses can be satisfied by setting q to true and r to false) and, after the operation, every variable appears the same number of time positively as negatively (including the new variables q, r).

Third, copy each clause three times. For each variable x, replace all occurrences of the variable x with a cycle of implications in the following way. Assume that x appears $6d$ times (the number of appearances is divisible by six due to the modifications in the previous paragraph and the copying step). Introduce new variables x_i for $1 \leq i \leq 3d$, y_i for $1 \leq i \leq d$ and clauses $(\neg x_i \vee x_{i+1} \vee y_{\lceil i/3 \rceil})$ and $(\neg x_i \vee x_{i+1} \vee \neg y_{\lceil i/3 \rceil})$ for $1 \leq i \leq 3d$ (with $x_{3d+1} = x_1$). Moreover, replace each occurrence of the variable x with one of the variables x_i in such a way that each variable x_i is used once in a positive literal and once in a negative one. In this manner each variable x_i and y_i is used exactly three times in a positive literal and three times in a negative one. Moreover, the new clauses form an implication cycle $x_1 \Rightarrow x_2 \Rightarrow \ldots \Rightarrow x_{3d} \Rightarrow x_1$, ensuring that all the variables x_i have equal value in any satisfying assignment of the formula.

Fourth, to make n' divisible by p we first copy the entire formula three times, creating a new set of variables for each copy. In this way we ensure that the number of variables is divisible by three. Then we add new variables in triples to make the number of variables divisible by p; note that since the number of variables is divisible by 3, there exists a number b, $0 \leq b < p$, such that after introducing b triples of variables the total number of variables will be divisible by p. For each triple x, y, z of new variables, we introduce six new clauses: all possible clauses that contain one literal for each variable x, y and z except for $(x \vee y \vee z)$ and $(\neg x \vee \neg y \vee \neg z)$. Note that the new clauses are easily satisfied by setting all new variables to true, while all new variables appear exactly three times positively and three times negatively. Moreover, as initially $p \leq n$, this step increases the size of the formula only by a constant multiplicative factor.

Finally, to achieve (d) and (e) take $\Phi' = \Phi \wedge \overline{\Phi}$, where $\overline{\Phi}$ is a copy of Φ on a disjoint copy of the variable set and with all literals reversed, i.e., positive occurrences are replaced by negative ones and vice versa. Of course, if Φ' is satisfiable then Φ as well, while if Φ is satisfiable, then we can copy the assignment to the copies of variables and reverse it, thus obtaining a feasible assignment for Φ'. Recall that before this step the number of variables was divisible by p. We can now partition the variable set into p parts, such that whenever we include a variable into one part, we include its copy in the same part as well. In order to prove that the property (e) holds, take any feasible solution to Φ', truncate the evaluation to $\mathrm{Vars}(\Phi)$ and copy it while reversing on $\overline{\Phi}$. □

8.4.2 Construction

In this section we show how to compute the graph G and the integer k' from the formula Φ' given by Lemma 130. As Lemma 130 increases the size of the formula by a constant multiplicative factor, we have that $n', m' = \mathcal{O}(\sqrt{pk})$ and $|\mathrm{Vars}^r| = n'/p = \mathcal{O}(\sqrt{k/p})$ for $1 \leq r \leq p$. Since each variable of Φ' appears in exactly 6 clauses, and each clause contains 3 literals, it follows that $m' = 2n'$.

Let $L = 1000 \cdot \left(1 + \frac{n'}{p}\right) = \mathcal{O}(\sqrt{k/p})$. For each part Vars^r, $1 \leq r \leq p$, we create six cliques Q^r_α, $1 \leq \alpha \leq 6$, each of size L. Let $\mathcal{Q}$ be the set of all the vertices of all cliques Q^r_α. In this manner we have $6p$ cliques. Intuitively, if we seek for a $6p$-cluster graph close to G, then the cliques are large enough so that merging two cliques is expensive — in the intended solution we have exactly one clique in each cluster.

For every variable $x \in \mathrm{Vars}^r$ create six vertices $w^x_{1,2}, w^x_{2,3}, \ldots, w^x_{5,6}, w^x_{6,1}$. Connect them into a cycle in this order; this cycle is called a 6-*cycle for the variable* x. Moreover, for each $1 \leq \alpha \leq 6$ and $v \in V(Q^r_\alpha)$, create edges $vw^x_{\alpha-1,\alpha}$ and $vw^x_{\alpha,\alpha+1}$ (we assume that the indices behave cyclically, i.e., $w^x_{6,7} = w^x_{6,1}$, $Q^r_7 = Q^r_1$ etc.). Let $\mathcal{W}$ be the set of all vertices $w^x_{\alpha,\alpha+1}$ for all variables x. Intuitively, the cheapest way to cut the 6-cycle for variable x is to assign the vertices $w^x_{\alpha,\alpha+1}$, $1 \leq \alpha \leq 6$ all either to the clusters with cliques with only odd indices or only with even indices. Choosing even indices corresponds to setting x to false, while choosing odd ones corresponds to setting x to true.

Let $r(x)$ be the index of the part that contains the variable x, that is, $x \in \mathrm{Vars}^{r(x)}$.

In each clause C we (arbitrarily) enumerate variables: for $1 \leq \eta \leq 3$, let $\mathrm{var}(C, \eta)$ be the variable in the η-th literal of C, and $\mathrm{sgn}(C, \eta) = 0$ if the η-th literal is negative and $\mathrm{sgn}(C, \eta) = 1$ otherwise.

For every clause C create nine vertices: $s^C_{\beta,\xi}$ for $1 \leq \beta, \xi \leq 3$. The edges incident to the vertex $s^C_{\beta,\xi}$ are defined as follows:

- for each $1 \leq \eta \leq 3$ create an edge $s^C_{\beta,\xi} w^{\mathrm{var}(C,\eta)}_{2\beta+2\eta-3,2\beta+2\eta-2}$;
- if $\xi = 1$, then for each $1 \leq \eta \leq 3$ connect $s^C_{\beta,\xi}$ to all vertices of one of the cliques the vertex $w^{\mathrm{var}(C,\eta)}_{2\beta+2\eta-3,2\beta+2\eta-2}$ is adjacent to depending on the sign of the η-th literal in C, that is, the clique $Q^{r(\mathrm{var}(C,\eta))}_{2\beta+2\eta-2-\mathrm{sgn}(C,\eta)}$;
- if $\xi > 1$, then for each $1 \leq \eta \leq 3$ connect $s^C_{\beta,\xi}$ to all vertices of both cliques the vertex $w^{\mathrm{var}(C,\eta)}_{2\beta+2\eta-3,2\beta+2\eta-2}$ is adjacent to, that is, the cliques $Q^{r(\mathrm{var}(C,\eta))}_{2\beta+2\eta-3}$ and $Q^{r(\mathrm{var}(C,\eta))}_{2\beta+2\eta-2}$.

We note that for a fixed vertex $s^C_{\beta,\xi}$, the aforementioned cliques $s^C_{\beta,\xi}$ is adjacent to are pairwise different, and they have pairwise different subscripts (but may have equal superscripts, i.e., belong to the same part). See Figure 8.1 for an illustration.

Let $\mathcal{S}$ be the set of all vertices $s^C_{\beta,\xi}$ for all clauses C. If we seek a $6p$-cluster graph close to the graph G, it is reasonable to put a vertex $s^C_{\beta,\xi}$ in a cluster together with one of the cliques this vertex is attached to. If $s^C_{\beta,\xi}$ is put in a cluster together with one of the vertices $w^{\mathrm{var}(C,\eta)}_{2\beta+2\eta-3,2\beta+2\eta-2}$ for $1 \leq \eta \leq 3$, we do not need to cut the appropriate edge. The vertices $s^C_{\beta,1}$ verify the assignment encoded by the variable vertices $w^x_{\alpha,\alpha+1}$; the vertices $s^C_{\beta,2}$ and $s^C_{\beta,3}$ help us to make all clusters of equal size (which is helpful in the soundness argument).

We note that $|V(G)| = 6pL + \mathcal{O}(n' + m') = \mathcal{O}(\sqrt{pk})$.

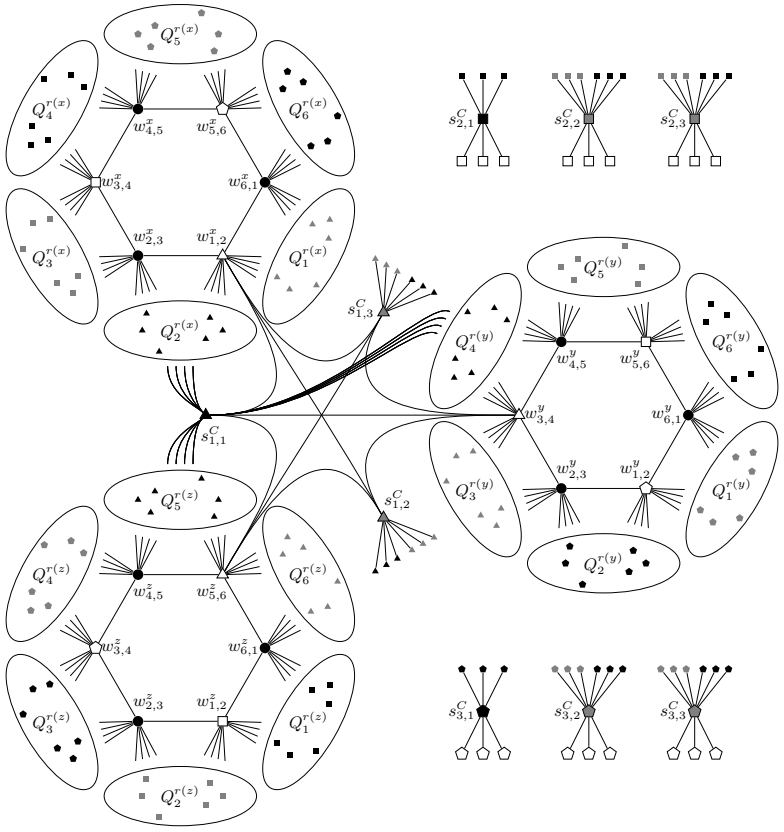

Figure 8.1: A part of the graph G created for the clause $C = (\neg x \vee \neg y \vee z)$, with $\mathrm{var}(C,1) = x$, $\mathrm{var}(C,2) = y$ and $\mathrm{var}(C,3) = z$. Note that the parts $r(x)$, $r(y)$ and $r(z)$ may be not be pairwise distinct. However, due to the rotation index β, in any case for a fixed vertex $s^C_{\beta,\xi}$ the cliques this vertex is adjacent to on this figure are pairwise distinct and have pairwise distinct subscripts.

We now define the budget k' for edge editions. To make the presentation more clear, we split this budget into few summands. Let

$$k_{\mathcal{Q}-\mathcal{Q}} = 0, \qquad k_{\mathcal{Q}-\mathcal{WS}} = (6n' + 36m')L, \qquad k^{\mathrm{all}}_{\mathcal{WS}-\mathcal{WS}} = 6p\binom{\frac{6n'+9m'}{6p}}{2},$$
$$k^{\mathrm{exist}}_{\mathcal{WS}-\mathcal{WS}} = 6n' + 27m', \qquad k^{\mathrm{save}}_{\mathcal{W}-\mathcal{W}} = 3n', \qquad k^{\mathrm{save}}_{\mathcal{W}-\mathcal{S}} = 9m',$$

and finally

$$k' = k_{\mathcal{Q}-\mathcal{Q}} + k_{\mathcal{Q}-\mathcal{WS}} + k^{\mathrm{all}}_{\mathcal{WS}-\mathcal{WS}} + k^{\mathrm{exist}}_{\mathcal{WS}-\mathcal{WS}} - 2k^{\mathrm{save}}_{\mathcal{W}-\mathcal{W}} - 2k^{\mathrm{save}}_{\mathcal{W}-\mathcal{S}}.$$

Note that since $p \leq k$, $L = \mathcal{O}(\sqrt{k/p})$ and $n', m' = \mathcal{O}(\sqrt{pk})$, we have $k' = \mathcal{O}(k)$.

The intuition behind this split is as follows. The intended solution for the p-Cluster Editing instance $(G, 6p, k')$ creates no edges between the cliques Q^r_α, each clique is contained in its own cluster, and $k_{\mathcal{Q}-\mathcal{Q}} = 0$. For each $v \in \mathcal{W} \cup \mathcal{S}$, the vertex v is assigned to a cluster with one clique v is adjacent to; $k_{\mathcal{Q}-\mathcal{WS}}$ accumulates the cost of removal of other edges in $E(\mathcal{Q}, \mathcal{W} \cup \mathcal{S})$. Finally, we count the editions in $(\mathcal{W} \cup \mathcal{S}) \times (\mathcal{W} \cup \mathcal{S})$ in an indirect way. First we cut all edges of

$E(\mathcal{W}\cup\mathcal{S},\mathcal{W}\cup\mathcal{S})$ (summand $k^{\text{exist}}_{\mathcal{WS}-\mathcal{WS}}$). We group the vertices of $\mathcal{W}\cup\mathcal{S}$ into clusters and add edges between vertices in each cluster; the summand $k^{\text{all}}_{\mathcal{WS}-\mathcal{WS}}$ corresponds to the cost of this operation when all the clusters are of the same size (and the number of edges is minimum possible). Finally, in summands $k^{\text{save}}_{\mathcal{W}-\mathcal{W}}$ and $k^{\text{save}}_{\mathcal{W}-\mathcal{S}}$ we count how many edges are removed and then added again in this process: $k^{\text{save}}_{\mathcal{W}-\mathcal{W}}$ corresponds to saving three edges from each 6-cycle in $E(\mathcal{W},\mathcal{W})$ and $k^{\text{save}}_{\mathcal{W}-\mathcal{S}}$ corresponds to saving one edge in $E(\mathcal{W},\mathcal{S})$ per each vertex $s^C_{\beta,\xi}$.

8.4.3 Completeness

We now show how to translate a satisfying assignment of the input formula Φ into a $6p$-cluster graph close to G.

Lemma 131. *If the input formula Φ is satisfiable, then there exists a $6p$-cluster graph G_0 on vertex set $V(G)$ such that $\mathcal{H}(G,G_0)=k'$.*

Proof. Let ϕ' be a satisfying assignment of the formula Φ' as guaranteed by Lemma 130. Recall that in each part Vars^r, the assignment ϕ' sets the same number of variables to true as to false.

To simplify the presentation, we identify the range of ϕ' with integers: $\phi'(x)=0$ if x is evaluated to false in ϕ' and $\phi'(x)=1$ otherwise. Moreover, for a clause C by $\eta(C)$ we denote the index of an arbitrarily chosen literal that satisfies C in the assignment ϕ'.

We create $6p$ clusters K^r_α, $1\le r\le p$, $1\le\alpha\le 6$, as follows:

- $Q^r_\alpha\subseteq K^r_\alpha$ for $1\le r\le p$, $1\le\alpha\le 6$;
- for $x\in\text{Vars}(\Phi')$, if $\phi'(x)=1$ then $w^x_{6,1},w^x_{1,2}\in K^{r(x)}_1$, $w^x_{2,3},w^x_{3,4}\in K^{r(x)}_3$, $w^x_{4,5},w^x_{5,6}\in K^{r(x)}_5$;
- for $x\in\text{Vars}(\Phi')$, if $\phi'(x)=0$ then $w^x_{1,2},w^x_{2,3}\in K^{r(x)}_2$, $w^x_{3,4},w^x_{4,5}\in K^{r(x)}_4$, $w^x_{5,6},w^x_{6,1}\in K^{r(x)}_6$;
- for each clause C of Φ' and each $1\le\beta,\xi\le 3$ we assign the vertex $s^C_{\beta,\xi}$ to the cluster $K^{r(\text{var}(C,\eta))}_{2\beta+2\eta-2-\phi'(\text{var}(C,\eta))}$, where $\eta=\eta(C)+\xi-1$.

Note that in this way $s^C_{\beta,\xi}$ belongs to the same cluster as its neighbor $w^{\text{var}(C,\eta)}_{2\beta+2\eta-3,2\beta+2\eta-2}$, where $\eta=\eta(C)+\xi-1$. See Figure 8.2 for an illustration.

Let us now compute $\mathcal{H}(G,G_0)$. We do not need to add nor delete any edges in $G[\mathcal{Q}]$. We note that each vertex $v\in\mathcal{W}\cup\mathcal{S}$ is assigned to a cluster with one clique Q^r_α it is adjacent to. Indeed, this is only non-trivial for vertices $s^C_{\beta,1}$ for clauses C and $1\le\beta\le 3$. Note, however, that $s^C_{\beta,1}$ is assigned to a cluster with the clique $Q^{r(\text{var}(C,\eta(C)))}_{2\beta+2\eta(C)-2-\phi'(\text{var}(C,\eta(C)))}$ and is adjacent to the clique $Q^{r(\text{var}(C,\eta(C)))}_{2\beta+2\eta(C)-2-\text{sgn}(\text{var}(C,\eta(C)))}$, but since literal with variable $\text{var}(C,\eta(C))$ satisfies C in the assignment ϕ', it follows that $\phi'(\text{var}(C,\eta(C)))=\text{sgn}(\text{var}(C,\eta(C)))$.

Therefore we need to cut $k_{\mathcal{Q}-\mathcal{WS}}=(6n'+36m')L$ edges in $E(\mathcal{Q},\mathcal{W}\cup\mathcal{S})$: L edges adjacent to each vertex $w^x_{\alpha,\alpha+1}$, $2L$ edges adjacent to each vertex $s^C_{\beta,1}$, and $5L$ edges adjacent to each vertex $s^C_{\beta,2}$ and $s^C_{\beta,3}$. We do not add any new edges between $\mathcal{Q}$ and $\mathcal{W}\cup\mathcal{S}$.

To count the number of editions in $G[\mathcal{W}\cup\mathcal{S}]$, let us first verify that the clusters K^r_α are of equal sizes. Fix cluster K^r_α, $1\le\alpha\le 6$, $1\le r\le p$. K^r_α contains two vertices $w^x_{\alpha-1,\alpha}$ and $w^x_{\alpha,\alpha+1}$ for each variable x with $\phi'(x)+\alpha$ being even. Since ϕ' evaluates the same number of variables in Vars^r to true as to false, we infer that each cluster K^r_α contains exactly n'/p vertices from $\mathcal{W}$, corresponding to $n'/(2p)=|\text{Vars}^r|/2$ variables of Vars^r.

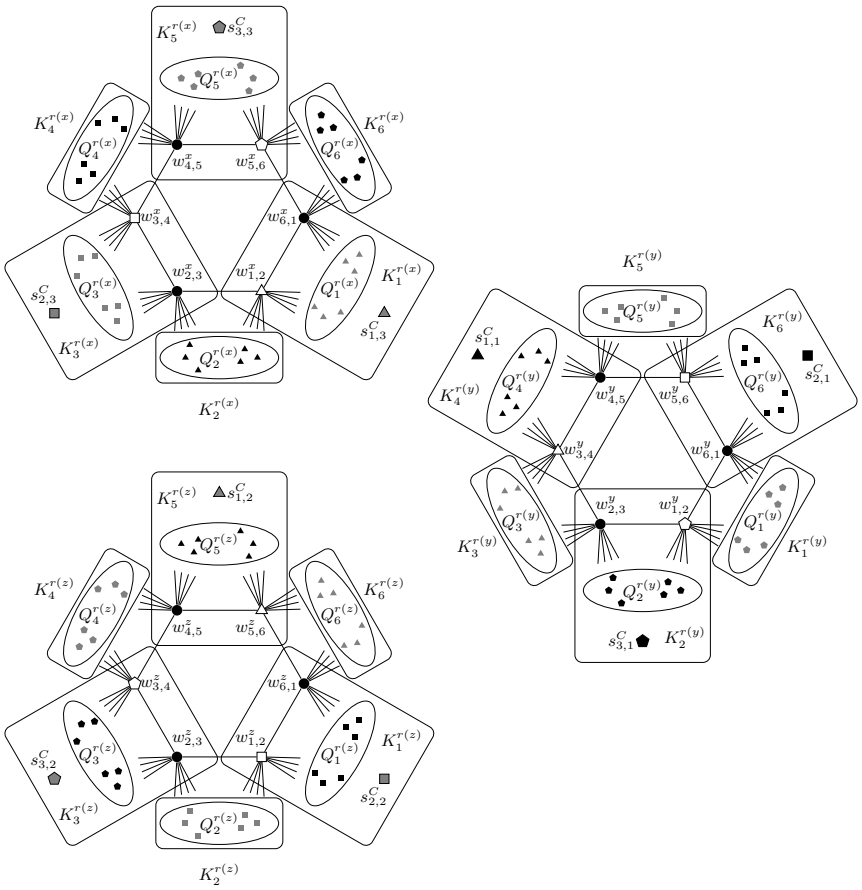

Figure 8.2: Parts of clusters for variables x, y and z with $\phi'(x) = 1$, $\phi'(y) = 0$, $\phi'(z) = 1$, and a clause $C = (\neg x \vee \neg y \vee z)$ with $\mathrm{var}(C, 1) = x$, $\mathrm{var}(C, 2) = y$, $\mathrm{var}(C, 3) = z$ and $\eta(C) = 2$ (note that both y and z satisfy C in the assignment ϕ', but y was chosen as a representative).

Now we need to verify that each cluster K_α^r contains the same number of vertices from $\mathcal{S}$. Let f be a function that maps the vertices of $\mathcal{S}$ to clusters they are belonging to. Recall that $f(s_{\beta,\xi}^C) = K_{2\beta+2\eta-2-\phi'(\mathrm{var}(C,\eta))}^{r(\mathrm{var}(C,\eta))}$, where $\eta = \eta(C) + \xi - 1$. We now claim the following.

Claim 1. *Fix a part r, $1 \leq r \leq p$, and an index α, $1 \leq \alpha \leq 6$. Then to each pair (x, C), where x is a variable with $r(x) = r$ present in a clause C, and $\phi'(x) = \alpha \bmod 2$, one can assign a pair of indices (β, ξ) with $1 \leq \beta, \xi \leq 3$ such that $f(s_{\beta,\xi}^C) = K_\alpha^r$. Moreover, this assignment can be constructed in such a manner that for all pairs (x, C) the corresponding vertices $s_{\beta,\xi}^C$ are pairwise different.*

Proof. Let η be such that $x = \mathrm{var}(C, \eta)$. Define indices (β, ξ) as follows:

$$\begin{aligned} \xi &= \eta - \eta(C) + 1, \\ \beta &= \frac{\alpha + \phi'(x)}{2} - \eta + 1, \end{aligned}$$

where the arithmetic operations are defined cyclically in a natural manner. Note that β is an

integer since $\phi'(x) = \alpha \bmod 2$. From these equalities it follows that:

$$\begin{aligned} \eta &= \eta(C) + \xi - 1, \\ \alpha &= 2\beta + 2\eta - 2 - \phi'(\mathrm{var}(C, \eta)), \end{aligned}$$

and so $f(s^C_{\beta,\xi}) = K^r_\alpha$. We are left with proving that triples (C, β, ξ) are pairwise different for different pairs (x, C). However, note that for different variables x appearing in C we have different indices η, and so the defined indices ξ will be different. ⌟

Observe that for a fixed part r and index α, there are exactly $3|\mathrm{Vars}^r| = 3n'/p$ pairs (x, C) satisfying the assumption in Claim 1: there are $|\mathrm{Vars}^r|/2$ variables in Vars^r that are evaluated to $\alpha \bmod 2$ in ϕ', and each variable appears in 6 different clauses. Thus, Claim 1 ensures that the preimage under f of each cluster is of cardinality at least $3n'/p$. Since there are $6p$ clusters in total, we infer that the union of these preimages is of cardinality at least $18n' = 9m'$ (recall that $m' = 2n'$). However, we have that $|\mathcal{S}| = 9m'$. Consequently, it follows that the preimage under f of each cluster is of size exactly $3n'/p$, so each cluster contains the same number of vertices from $\mathcal{S}$.

We now count the number of editions in $G[\mathcal{W} \cup \mathcal{S}]$ as sketched in the construction section. The subgraph $G[\mathcal{W} \cup \mathcal{S}]$ contains $6n' + 27m'$ edges: one 6-cycle for each variable and three edges incident to each of the nine vertices $s^C_{\beta,\xi}$ for each clause C. Each cluster K^r_α contains n'/p vertices from $\mathcal{W}$ and $\frac{3m'}{2p}$ vertices from $\mathcal{S}$. If we deleted all edges in $G[\mathcal{W} \cup \mathcal{S}]$ and then added all the missing edges in the clusters, we would make $k^{\mathrm{exist}}_{\mathcal{WS}-\mathcal{WS}} + k^{\mathrm{all}}_{\mathcal{WS}-\mathcal{WS}}$ editions, due to the clusters being equal-sized. However, in this manner we sometimes delete an edge and then introduce it again; thus, for each edge of $G[\mathcal{W} \cup \mathcal{S}]$ that is contained in one cluster K^r_α, we should subtract 2 in this counting scheme.

For each variable x, exactly three edges of the form $w^x_{\alpha-1,\alpha} w^x_{\alpha,\alpha+1}$ are contained in one cluster; this gives a total of $k^{\mathrm{save}}_{\mathcal{W}-\mathcal{W}} = 3n'$ saved edges. For each clause C each vertex $s^C_{\beta,\xi}$ is assigned to a cluster with one of the vertices $w^{\mathrm{var}(C,\eta)}_{2\beta+2\eta-3,2\beta+2\eta-2}$, $1 \leq \eta \leq 3$, thus exactly one of the edges incident to $s^C_{\beta,\xi}$ is contained in one cluster. This sums up to $k^{\mathrm{save}}_{\mathcal{W}-\mathcal{S}} = 9m'$ saved edges, and we infer that the $6p$-cluster graph G_0 can be obtained from G by exactly $k' = k_{\mathcal{Q}-\mathcal{Q}} + k_{\mathcal{Q}-\mathcal{WS}} + k^{\mathrm{exist}}_{\mathcal{WS}-\mathcal{WS}} + k^{\mathrm{all}}_{\mathcal{WS}-\mathcal{WS}} - 2k^{\mathrm{save}}_{\mathcal{W}-\mathcal{W}} - 2k^{\mathrm{save}}_{\mathcal{W}-\mathcal{S}}$ editions. □

8.4.4 Soundness

We need the following simple bound on the number of edges of a cluster graph.

Lemma 132. *Let a, b be positive integers and H be a cluster graph with ab vertices and at most a clusters. Then $|E(H)| \geq a\binom{b}{2}$ and equality holds if and only if H is an a-cluster graph and each cluster of H has size exactly b.*

Proof. It suffices to note that if not all clusters of H are of size b, there is one of size at least $b+1$ and one of size at most $b-1$ or the number of clusters is less than a; then, moving a vertex from the largest cluster of H to a new or the smallest cluster strictly decreases the number of edges of H. □

We are ready to show how to translate a p'-cluster graph G_0 with $p' \leq 6p$ and $\mathcal{H}(G_0, G) \leq k'$, into a satisfying assignment of the input formula Φ.

Lemma 133. *If there exists a p'-cluster graph G_0 with $V(G) = V(G_0)$, $p' \leq 6p$, and $\mathcal{H}(G, G_0) \leq k'$, then the formula Φ is satisfiable.*

Proof. By Lemma 121, we may assume that each clique Q^r_α is contained in one cluster in G_0. Let $F = E(G_0) \triangle E(G)$ be the editing set, $|F| \leq k'$.

Before we start, we present some intuition. The cluster graph G_0 may differ from the one constructed in the completeness step in two significant ways, both leading to some savings in the edges incident to $\mathcal{W} \cup \mathcal{S}$ that may not be included in F. First, it may not be true that each cluster contains exactly one clique Q^r_α. However, since the number of cliques is at most $6p$, this may happen only if some clusters contain more than one clique Q^r_α, and we need to add L^2 edges to merge each pair of cliques that belong to the same cluster. Second, a vertex $v \in \mathcal{W} \cup \mathcal{S}$ may not be contained in a cluster together with one of the cliques it is adjacent to. However, as each such vertex needs to be separated from *all* its adjacent cliques (compared to *all but one* in the completeness step), this costs us additional L edges to remove. The large multiplicative constant in the definition of L ensures us that in both these ways we pay more than we save on the edges incident with $\mathcal{W} \cup \mathcal{S}$. We now proceed to the formal argumentation.

We define the following quantities.

$$\ell_{\mathcal{Q}-\mathcal{Q}} = |F \cap (\mathcal{Q} \times \mathcal{Q})|, \qquad \ell_{\mathcal{Q}-\mathcal{WS}} = |F \cap E_G(\mathcal{Q}, \mathcal{W} \cup \mathcal{S})|,$$
$$\ell^{\mathrm{all}}_{\mathcal{WS}-\mathcal{WS}} = |E(G_0[\mathcal{W} \cup \mathcal{S}])|,$$
$$\ell^{\mathrm{save}}_{\mathcal{W}-\mathcal{W}} = |E_G(\mathcal{W}, \mathcal{W}) \cap E_{G_0}(\mathcal{W}, \mathcal{W})| \qquad \ell^{\mathrm{save}}_{\mathcal{W}-\mathcal{S}} = |E_G(\mathcal{W}, \mathcal{S}) \cap E_{G_0}(\mathcal{W}, \mathcal{S})|.$$

Recall that $k^{\mathrm{exist}}_{\mathcal{WS}-\mathcal{WS}} = |E(G[\mathcal{W} \cup \mathcal{S}])| = 6n' + 27m'$. Similarly as in the completeness proof, we have that

$$|F| \geq \ell_{\mathcal{Q}-\mathcal{Q}} + \ell_{\mathcal{Q}-\mathcal{WS}} + \ell^{\mathrm{all}}_{\mathcal{WS}-\mathcal{WS}} + k^{\mathrm{exist}}_{\mathcal{WS}-\mathcal{WS}} - 2\ell^{\mathrm{save}}_{\mathcal{W}-\mathcal{W}} - 2\ell^{\mathrm{save}}_{\mathcal{W}-\mathcal{S}}.$$

Indeed, $\ell_{\mathcal{Q}-\mathcal{Q}}$ and $\ell_{\mathcal{Q}-\mathcal{WS}}$ count (possibly not all) edges of F that are incident to the vertices of $\mathcal{Q}$. The edges of $F \cap ((\mathcal{W} \cup \mathcal{S}) \times (\mathcal{W} \cup \mathcal{S}))$ are counted in an indirect way: each edge of $G[\mathcal{W} \cup \mathcal{S}]$ is deleted ($k^{\mathrm{exist}}_{\mathcal{WS}-\mathcal{WS}}$) and each edge of $G_0[\mathcal{W} \cup \mathcal{S}]$ is added ($\ell^{\mathrm{all}}_{\mathcal{WS}-\mathcal{WS}}$). Then, the edges that are counted twice in this manner are subtracted ($\ell^{\mathrm{save}}_{\mathcal{W}-\mathcal{W}}$ and $\ell^{\mathrm{save}}_{\mathcal{W}-\mathcal{S}}$).

We say that a cluster is *crowded* if it contains at least two cliques Q^r_α and *proper* if it contains exactly one clique Q^r_α. A clique Q^r_α that is contained in a crowded (proper) cluster is called a *crowded* (*proper*) clique.

Let a be the number of crowded cliques. Note that $\ell_{\mathcal{Q}-\mathcal{Q}} - k_{\mathcal{Q}-\mathcal{Q}} = |F \cap (\mathcal{Q} \times \mathcal{Q})| - 0 \geq aL^2/2$, since each vertex in a crowded clique needs to be connected to at least one other crowded clique.

We say that a vertex $v \in \mathcal{W} \cup \mathcal{S}$ is *attached* to a clique Q^r_α, if it is adjacent to all vertices of the clique in G. Moreover, we say that a vertex $v \in \mathcal{W} \cup \mathcal{S}$ is *alone* if it is contained in a cluster in G_0 that does not contain any clique v is attached to. Let n^{alone} be the number of alone vertices.

Let us now count the number of vertices a fixed clique Q^r_α is attached to. Recall that $|\mathrm{Vars}^r| = n'/p$. For each variable $x \in \mathrm{Vars}^r$ the clique Q^r_α is attached to two vertices $w^x_{\alpha-1,\alpha}$ and $w^x_{\alpha,\alpha+1}$. Moreover, each variable $x \in \mathrm{Vars}^r$ appears in exactly six clauses: thrice positively and thrice negatively. For each such clause C, Q^r_α is attached to the vertex $s^C_{\beta,2}$ for exactly one choice of the value $1 \leq \beta \leq 3$ and to the vertex $s^C_{\beta,3}$ for exactly one choice of the value $1 \leq \beta \leq 3$. Moreover, if x appears in C positively and α is odd, or if x appears in C negatively and α is even, then Q^r_α is attached to the vertex $s^C_{\beta,1}$ for exactly one choice of the value $1 \leq \beta \leq 3$. We infer that the clique Q^r_α is attached to exactly fifteen vertices from $\mathcal{S}$ for each variable $x \in \mathrm{Vars}^r$. Therefore, there are exactly $17|\mathrm{Vars}^r| = 17n'/p$ vertices of $\mathcal{W} \cup \mathcal{S}$ attached to Q^r_α: $2n'/p$ from $\mathcal{W}$ and $15n'/p$ from $\mathcal{S}$.

Take an arbitrary vertex $v \in \mathcal{W} \cup \mathcal{S}$ and assume that v is attached to b_v cliques, and a_v out of them are crowded. As F needs to contain all edges of G that connect v with cliques that belong

to a different cluster than v, we infer that $|F \cap E_G(\{v\}, \mathcal{Q})| \geq (b_v - \max(1, a_v))L$. Moreover, if v is alone, $|F \cap E_G(\{v\}, \mathcal{Q})| \geq b_v L \geq 1 \cdot L + (b_v - \max(1, a_v))L$. Hence

$$\begin{aligned}\ell_{\mathcal{Q}-\mathcal{WS}} = |F \cap E_G(\mathcal{Q}, \mathcal{W} \cup \mathcal{S})| &\geq n^{\text{alone}} L + \sum_{v \in \mathcal{W} \cup \mathcal{S}} (b_v - \max(1, a_v))L \\ &\geq n^{\text{alone}} L + \sum_{v \in \mathcal{W} \cup \mathcal{S}} (b_v - 1)L - \sum_{v \in \mathcal{W} \cup \mathcal{S}} a_v L.\end{aligned}$$

Recall that $\sum_{v \in \mathcal{W} \cup \mathcal{S}} (b_v - 1)L = k_{\mathcal{Q}-\mathcal{WS}}$. Therefore, using the fact that each clique is attached to exactly $17n'/p$ vertices of $\mathcal{W} \cup \mathcal{S}$, we obtain that

$$\begin{aligned}\ell_{\mathcal{Q}-\mathcal{WS}} - k_{\mathcal{Q}-\mathcal{WS}} &= |F \cap E_G(\mathcal{Q}, \mathcal{W} \cup \mathcal{S})| - k_{\mathcal{Q}-\mathcal{WS}} \\ &\geq n^{\text{alone}} L - \sum_{v \in \mathcal{W} \cup \mathcal{S}} a_v L = n^{\text{alone}} L - 17aLn'/p.\end{aligned}$$

In G_0, the vertices of $\mathcal{W} \cup \mathcal{S}$ are split between $p' \leq 6p$ clusters and there are $6n' + 9m'$ of them. By Lemma 132, the minimum number of edges of $G_0[\mathcal{W} \cup \mathcal{S}]$ is attained when all clusters are of equal size and the number of clusters is maximum possible. We infer that $\ell^{\text{all}}_{\mathcal{WS}-\mathcal{WS}} \geq k^{\text{all}}_{\mathcal{WS}-\mathcal{WS}}$.

We are left with bounding $\ell^{\text{save}}_{\mathcal{W}-\mathcal{W}}$ and $\ell^{\text{save}}_{\mathcal{W}-\mathcal{S}}$. Recall that $k^{\text{save}}_{\mathcal{W}-\mathcal{W}}$ counts three edges out of each 6-cycle constructed per variable of Φ', $|k^{\text{save}}_{\mathcal{W}-\mathcal{W}}| = 3n'$, whereas $k^{\text{save}}_{\mathcal{W}-\mathcal{S}}$ counts one edge per each vertex $s^C_{\beta,\xi} \in \mathcal{S}$, $k^{\text{save}}_{\mathcal{W}-\mathcal{S}} = 9m' = |\mathcal{S}|$.

Consider a crowded cluster K with $c > 1$ crowded cliques. We say that K *interferes* with a vertex $v \in \mathcal{W} \cup \mathcal{S}$ if v is attached to a clique in K. As each clique is attached to exactly $17n'/p$ vertices of $\mathcal{W} \cup \mathcal{S}$, $2n'/p$ belonging to $\mathcal{W}$ and $15n'/p$ to $\mathcal{S}$, in total at most $2an'/p$ vertices of $\mathcal{W}$ and at most $15an'/p$ vertices of $\mathcal{S}$ interfere with a crowded cluster.

Fix a variable $x \in \text{Vars}(\Phi')$. If none of the vertices $w^x_{\alpha,\alpha+1} \in \mathcal{W}$ interferes with any crowded cluster K, then all the cliques $Q^{r(x)}_{\alpha'}$, $1 \leq \alpha' \leq 6$, are proper cliques, each contained in a different cluster in G_0. Moreover, if additionally no vertex $w^x_{\alpha,\alpha+1}$, $1 \leq \alpha \leq 6$, is alone, then in the 6-cycle constructed for the variable x at most three edges are not in F. On the other hand, if some of the vertices $w^x_{\alpha,\alpha+1} \in \mathcal{W}$ interfere with a crowded cluster K, or at least one of them is alone, it may happen that all six edges of this 6-cycle are contained in one cluster of G_0. The total number of 6-cycles that contain either alone vertices or vertices interfering with crowded clusters is bounded by $n^{\text{alone}} + an'/p$, as every clique is attached to exactly n'/p 6-cycles. In $k^{\text{save}}_{\mathcal{W}-\mathcal{W}}$ we counted three edges per a 6-cycle, while in $\ell^{\text{save}}_{\mathcal{W}-\mathcal{W}}$ we counted at most three edges per every 6-cycles except 6-cycles that either contain alone vertices or vertices attached to crowded cliques, for which we counted at most six edges. Hence, we infer that

$$\ell^{\text{save}}_{\mathcal{W}-\mathcal{W}} - k^{\text{save}}_{\mathcal{W}-\mathcal{W}} \leq 3(n^{\text{alone}} + an'/p).$$

We claim that if a vertex $s^C_{\beta,\xi} \in \mathcal{S}$ (i) is not alone, and (ii) is not attached to a crowded clique, and (iii) is not adjacent to any alone vertex in $\mathcal{W}$, then at most one edge from $E(\{s^C_{\beta,\xi}\}, \mathcal{W})$ may not be in F. Recall that $s^C_{\beta,\xi}$ has exactly three neighbors in $\mathcal{W}$, each of them attached to exactly two cliques and all these six cliques are pairwise distinct; moreover, $s^C_{\beta,\xi}$ is attached only to these six cliques, if $\beta = 2, 3$, or only to three out of these six, if $\beta = 1$. Observe that (i) and (ii) imply that $s^C_{\beta,\xi}$ is in the same cluster as exactly one of the six cliques attached to his neighbors in $\mathcal{W}$, so if it was in the same cluster as two of his neighbors in $\mathcal{W}$, then at least one of them would be

alone, contradicting (iii). Therefore, if conditions (i), (ii), and (iii) are satisfied, then at most one edge adjacent to $s^C_{\beta,\xi}$ may be not contained in F. However, if at least one of (i), (ii) or (iii) is not satisfied, then all three edges incident to $s^S_{\beta,\xi}$ may be contained in one cluster. As each vertex in $\mathcal{W}$ is adjacent to at most 18 vertices in $\mathcal{S}$ (at most 3 per every clause in which the variable is present), there are at most $18n^{\text{alone}}$ vertices $s^C_{\beta,\xi}$ that are alone or adjacent to an alone vertex in $\mathcal{W}$. Note also that the number of vertices of $\mathcal{S}$ interfering with crowded clusters is bounded by $15an'/p$, as each of a crowded cliques has exactly $15n'/p$ vertices of $\mathcal{S}$ attached. Thus, we are able to bound the number of vertices of $\mathcal{S}$ for which (i), (ii) or (iii) does not hold by $18n^{\text{alone}}+15an'/p$. As in $k^{\text{save}}_{\mathcal{W}-\mathcal{S}}$ we counted one edge per every vertex of $\mathcal{S}$, while in $\ell^{\text{save}}_{\mathcal{W}-\mathcal{S}}$ we counted at most one edge per every vertex of $\mathcal{S}$ except vertices not satisfying (i), (ii), or (iii), for which we counted at most three edges, we infer that

$$\ell^{\text{save}}_{\mathcal{W}-\mathcal{S}} - k^{\text{save}}_{\mathcal{W}-\mathcal{S}} \leq 2(18n^{\text{alone}} + 15an'/p).$$

Summing up all the bounds:

$$\begin{aligned}
|F| - k' &\geq (\ell_{\mathcal{Q}-\mathcal{Q}} - k_{\mathcal{Q}-\mathcal{Q}}) + (\ell_{\mathcal{Q}-\mathcal{WS}} - k_{\mathcal{Q}-\mathcal{WS}}) + (\ell^{\text{all}}_{\mathcal{WS}-\mathcal{WS}} - k^{\text{all}}_{\mathcal{WS}-\mathcal{WS}}) \\
&\quad -2(\ell^{\text{save}}_{\mathcal{W}-\mathcal{W}} - k^{\text{save}}_{\mathcal{W}-\mathcal{W}}) - 2(\ell^{\text{save}}_{\mathcal{W}-\mathcal{S}} - k^{\text{save}}_{\mathcal{W}-\mathcal{S}}) \\
&\geq aL^2/2 + n^{\text{alone}}L - 17aLn'/p + 0 \\
&\quad -6(n^{\text{alone}} + an'/p) - 4(18n^{\text{alone}} + 15an'/p) \\
&\geq a + n^{\text{alone}} \geq 0.
\end{aligned}$$

The second to last inequality follows from the choice of the value of L, $L = 1000 \cdot \left(1 + \frac{n'}{p}\right)$; note that in particular $L \geq 1000$.

We infer that $a = 0$, that is, each clique Q^r_α is contained in a different cluster of G_0, and each cluster of G_0 contains exactly one such clique. Moreover, $n^{\text{alone}} = 0$, that is, each vertex $v \in \mathcal{W} \cup \mathcal{S}$ is contained in a cluster with at least one clique v is attached to; as all cliques are proper, v is contained in a cluster with exactly one clique v is attached to and $\ell_{\mathcal{Q}-\mathcal{WS}} = k_{\mathcal{Q}-\mathcal{WS}}$.

Recall that $|F \cap ((\mathcal{W} \cup \mathcal{S}) \times (\mathcal{W} \cup \mathcal{S}))| = \ell^{\text{all}}_{\mathcal{WS}-\mathcal{WS}} + k^{\text{exist}}_{\mathcal{WS}-\mathcal{WS}} - 2\ell^{\text{save}}_{\mathcal{W}-\mathcal{W}} - 2\ell^{\text{save}}_{\mathcal{W}-\mathcal{S}}$. As each clique is now proper and no vertex is alone, for each variable x at most three edges out of the 6-cycle $w^x_{\alpha,\alpha+1}$, $1 \leq \alpha \leq 6$, are not in F, that is, $\ell^{\text{save}}_{\mathcal{W}-\mathcal{W}} \leq k^{\text{save}}_{\mathcal{W}-\mathcal{W}}$. Moreover, for each vertex $s^C_{\beta,\xi} \in \mathcal{S}$, the three neighbors of $s^C_{\beta,\xi}$ are contained in different clusters and at most one edge incident to $s^C_{\beta,\xi}$ is not in F, that is, $\ell^{\text{save}}_{\mathcal{W}-\mathcal{S}} \leq k^{\text{save}}_{\mathcal{W}-\mathcal{S}}$. As $|F| \leq k'$ and $\ell^{\text{all}}_{\mathcal{WS}-\mathcal{WS}} \geq k^{\text{all}}_{\mathcal{WS}-\mathcal{WS}}$, these inequalities are tight: exactly three edges out of each 6-cycle are not in F, and exactly one edge adjacent to a vertex in $\mathcal{S}$ is not in F.

Consider an assignment ϕ' of $\text{Vars}(\Phi')$ that assigns $\phi'(x) = 1$ if the vertices $w^x_{\alpha,\alpha+1}$, $1 \leq \alpha \leq 6$ are contained in clusters with cliques $Q^{r(x)}_1$, $Q^{r(x)}_3$, and $Q^{r(x)}_5$ (i.e., the edges $w^x_{6,1}w^x_{1,2}$, $w^x_{2,3}w^x_{3,4}$ and $w^x_{4,5}w^x_{5,6}$ are not in F), and $\phi'(x) = 0$ otherwise (i.e., if the vertices $w^x_{\alpha,\alpha+1}$, $1 \leq \alpha \leq 6$ are contained in clusters with cliques $Q^{r(x)}_2$, $Q^{r(x)}_4$ and $Q^{r(x)}_6$) — a direct check shows that these are the only ways to save 3 edges inside a 6-cycle. We claim that ϕ' satisfies Φ'. Consider a clause C. The vertex $s^C_{1,1}$ is contained in a cluster with one of the three cliques it is attached to (as $n^{\text{alone}} = 0$), say $Q^r_{\alpha'}$, and with one of the three vertices of $\mathcal{W}$ it is adjacent to, say $w^x_{\alpha,\alpha+1}$. Therefore $r(x) = r$, $w^x_{\alpha,\alpha+1}$ is contained in the same cluster as $Q^r_{\alpha'}$, and it follows that the literal in C that contains x satisfies C in the assignment ϕ'. □

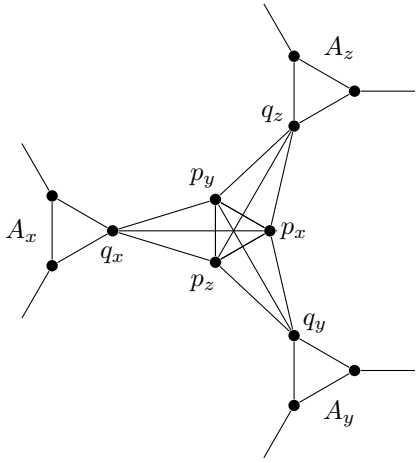

Figure 8.3: The gadget for a clause C with variables x, y and z.

8.5 General clustering under ETH

In this section we prove Theorem 119, namely that the Cluster Editing problem without restriction on the number of clusters in the output does not admit a $\mathcal{O}^*(2^{o(k)})$ algorithm unless the Exponential Time Hypothesis fails. The following lemma provides a linear reduction from the 3-CNF-SAT problem.

Lemma 134. *There exists a polynomial-time algorithm that, given a 3-CNF formula Φ with n variables and m clauses, constructs a* Cluster Editing *instance (G, k) such that (i) Φ is satisfiable if and only if (G, k) is a YES-instance, and (ii) $|V(G)| + |E(G)| + k = \mathcal{O}(n + m)$.*

Proof. By standard arguments (see for instance Lemma 130), we may assume that each clause of Φ consists of exactly three literals with different variables and each variable appears at least twice: at least once in a positive literal and at least once in a negative one. Let $\mathrm{Vars}(\Phi)$ denote the set of variables of Φ. For a variable x, let s_x be the number of appearances of x in the formula Φ. For a clause C with variables x, y, and z, we denote by $l_{x,C}$ the literal of C that contains x (i.e., $l_{x,C} = x$ or $l_{x,C} = \neg x$).

Construction. We construct a graph $G = (V, E)$ as follows. First, for each variable x we introduce a cycle A_x of length $4s_x$. For each clause C where x appears we assign four consecutive vertices $a^j_{x,C}$, $1 \leq j \leq 4$ on the cycle A_x. If the vertices assigned to a clause C' follow the vertices assigned to a clause C on the cycle A_x, we let $a^5_{x,C} = a^1_{x,C'}$.

Second, for each clause C with variables x, y, and z we introduce a gadget with 6 vertices $V_C = \{p_x, p_y, p_z, q_x, q_y, q_z\}$ with all inner edges except for q_xq_y, q_yq_z, and q_zq_x (see Figure 8.3). If $l_{x,C} = x$ then we connect q_x to the vertices $a^1_{x,C}$ and $a^2_{x,C}$, and if $l_{x,C} = \neg x$, we connect q_x to $a^2_{x,C}$ and $a^3_{x,C}$. We proceed analogously for variables y and z in the clause C. We set $k = 8m + 2\sum_{x \in \mathrm{Vars}(\Phi)} s_x = 14m$. This finishes the construction of the Cluster Editing instance (G, k). Clearly $|V(G)| + |E(G)| + k = \mathcal{O}(n + m)$. We now prove that (G, k) is a YES-instance if and only if Φ is satisfiable.

Completeness. Assume that Φ is satisfiable, and let ϕ be a satisfying assignment for Φ. We construct a set $F \subseteq V \times V$ as follows. First, for each variable x we take into F the edges $a^2_{x,C}a^3_{x,C}$, $a^4_{x,C}a^5_{x,C}$ for each clause C if $\phi(x)$ is true and the edges $a^1_{x,C}a^2_{x,C}$, $a^3_{x,C}a^4_{x,C}$ for each clause C otherwise. Second, let C be a clause of Φ with variables x, y, and z and, without loss of generality,

assume that the literal $l_{x,C}$ satisfies C in the assignment ϕ. For such a clause C we add to F eight elements: the edges q_xp_x, q_xp_y, q_xp_z, the four edges that connect q_y and q_z to the cycles A_y and A_z, and the non-edge q_yq_z.

Clearly $|F| = \sum_{x \in \mathrm{Vars}(\Phi)} 2s_x + 8m = k$. We now verify that $G \triangle F$ is a cluster graph. For each cycle A_x, the removal of the edges in F results in splitting the cycle into $2s_x$ two-vertex clusters. For each clause C with variables x, y, z, satisfied by the literal $l_{x,C}$ in the assignment ϕ, the vertices p_x, p_y, p_z, q_y, and q_z form a 5-vertex cluster. Moreover, since $l_{x,C}$ is true in ϕ, the edge that connects the two neighbors of q_x on the cycle A_x is not in F, so q_x and these two neighbors form a three-vertex cluster.

Soundness. Let F be a minimum size feasible solution to the Cluster Editing instance (G, k). By Lemma 121, for each clause C with variables x, y, and z, the vertices p_x, p_y, and p_z are contained in a single cluster in $G \triangle F$. Denote the vertex set of this cluster by Z_C; note that clusters Z_C for different clauses C are not necessarily different. For a minimum size feasible solution F, let $\Lambda(F)$ denote the set of clauses C such that cluster Z_C fully contained in the vertex set V_C. Without loss of generality we choose a minimum size feasible solution F such that $|\Lambda(F)|$ is maximum possible.

Informally, we are going to show that the solution F needs to look almost like the one constructed in the proof of completeness. The crucial observation is that if we want to create a six-vertex cluster $Z_C = V_C$ then we need to put nine (instead of eight) elements in F that are incident to V_C. Let us now proceed to the formal arguments.

Fix a variable x and let $F_x = F \cap (V(A_x) \times V(A_x))$. We claim that $|F_x| \geq 2s_x$ and, moreover, if $|F| = 2s_x$ then F_x consists of every second edge of the cycle A_x. Note that $A_x \triangle F_x$ is a cluster graph; assume that there are γ clusters in $A_x \triangle F_x$ with sizes α_j for $1 \leq j \leq \gamma$. If $\gamma = 1$ then, as $s_x \geq 2$,

$$|F_x| = |\alpha_1| = \binom{4s_x}{2} - 4s_x = 8s_x^2 - 6s_x > 2s_x.$$

Otherwise, in a cluster with α_j vertices we need to add at least $\binom{\alpha_j}{2} - (\alpha_j - 1)$ edges and remove at least two edges of A_x leaving the cluster. Using $\sum \alpha_j = 4s_x$, we infer that

$$|F_x| \geq \gamma + \sum_{j=1}^{\gamma} \left(\binom{\alpha_j}{2} - (\alpha_j - 1) \right) = \frac{1}{2} \sum_{j=1}^{\gamma} (\alpha_j^2 - 3\alpha_j + 4) = 2s_x + \frac{1}{2} \sum_{j=1}^{\gamma} (\alpha_j - 2)^2.$$

Thus $|F_x| \geq 2s_x$, and $|F_x| = 2s_x$ only if for all $1 \leq j \leq \gamma$ we have $\alpha_j = 2$ and in each two-vertex cluster of $A_x \triangle F_x$, F_x does not contain the edge in this cluster and contains two edges of A_x that leave this cluster. This situation occurs only if F_x consists of every second edge of the cycle A_x.

We now focus on a gadget for some clause C with variables x, y, and z. Let $F_C = F \cap (V_C \times (V_C \cup V(A_x) \cup V(A_y) \cup V(A_z)))$. We claim that $|F_C| \geq 8$ and there are very limited ways in which we can obtain $|F_C| = 8$.

Recall that the vertices p_x, p_y, and p_z are contained in a single cluster in $G \triangle F$ with vertex set Z_C. We now distinguish subcases, depending on how many of the vertices q_x, q_y, and q_z are in Z_C.

If $q_x, q_y, q_z \notin Z_C$, then $\{p_x, p_y, p_z\} \times \{q_x, q_y, q_x\} \subseteq F_C$ and $|F_C| \geq 9$.

If $q_x \in Z_C$, but $q_y, q_z \notin Z_C$, then $\{p_x, p_y, p_z\} \times \{q_y, q_z\} \subseteq F_C$. If there is a vertex $v \in Z_C \setminus V_C$, then F needs to contain three elements vp_x, vp_y, and vp_z. In this case F' constructed from F by replacing all elements incident to $\{q_x, p_x, p_y, p_z\}$ with all eight edges of G incident to this set is a feasible solution to (G, k) of size smaller than F, a contradiction to the assumption of the minimality of F. Thus, $Z_C = \{q_x, p_x, p_y, p_z\}$, and F_C contains the eight edges of G incident to Z_C.

If $q_x, q_y \in Z_C$ but $q_z \notin Z_C$, then $q_zp_x, q_zp_y, q_zp_z, q_xq_y \in F_C$. If there is a vertex $v \in Z_C \setminus V_C$, then F_C contains the three elements vp_x, vp_y, vp_z and at least one of the elements of $\{vq_x, vq_y\}$. In this case F' constructed from F by replacing all elements incident to $\{p_x, p_y, p_z, q_x, q_y\}$ with all seven edges of G incident to this set and the non-edge q_xq_y is a feasible solution to (G, k) of size not greater than F. From the construction it also follows that $\Lambda(F') \supsetneq \Lambda(F)$, which is a contradiction with the choice of F. Thus $Z_C = \{p_x, p_y, p_z, q_x, q_y\}$ and F_C contains all seven edges incident to Z_C and the non-edge q_xq_y.

In the last case we have that $V_C \subseteq Z_C$, and $q_xq_y, q_yq_z, q_zq_x \in F_C$. There are six edges connecting V_C and $V(A_x) \cup V(A_y) \cup V(A_z)$ in G, and all these edges are incident to different vertices of $V(A_x) \cup V(A_y) \cup V(A_z)$. Let uv be one of these six edges, $u \in V_C$, $v \notin V_C$. If $v \in Z_C$ then F contains five non-edges connecting v to $V_C \setminus \{u\}$. Otherwise, if $v \notin Z_C$ then F contains the edge uv. We infer that F_C contains at least six elements that have exactly one endpoint in V_C, and hence $|F_C| \geq 9$.

We now note that the sets F_C for all clauses C and the sets F_x for all variables x are pairwise disjoint. Recall that $|F_x| \geq 2s_x$ for any variable x and $|F_C| \geq 8$ for any clause C. As $|F| \leq 14m = 8m + \sum_x 2s_x$, we infer that $|F_x| = 2s_x$ for any variable x, $|F_C| = 8$ for any clause C and F contains no elements that are not in any set F_x or F_C.

As $|F_x| = 2s_x$ for each variable x, the set F_x consists of every second edge of the cycle A_x. We construct an assignment ϕ as follows: $\phi(x)$ is true if for all clauses C where x appears we have $a^2_{x,C}a^3_{x,C}, a^4_{x,C}a^5_{x,C} \in F$ and $\phi(x)$ is false if $a^1_{x,C}a^2_{x,C}, a^3_{x,C}a^4_{x,C} \in F$. We claim that ϕ satisfies Φ. Consider a clause C with variables x, y, and z. As $|F_C| = 8$, by the analysis above one of two situations occur: $|Z_C| = 4$, say $Z_C = \{p_x, p_y, p_z, q_x\}$, or $|Z_C| = 5$, say $Z_C = \{p_x, p_y, p_z, q_x, q_y\}$. In both cases, F_C consists only of all edges of G that connect Z_C with $V(G) \setminus Z_C$ and the non-edges of $G[Z_C]$. Thus, in both cases the two edges that connect q_z with the cycle A_z are not in F. Thus, the two neighbors of q_z on the cycle A_z are connected by an edge not in F, and it follows that the literal $l_{z,C}$ satisfies the clause C in the assignment ϕ. □

Lemma 134 directly implies the proof of Theorem 119.

Proof of Theorem 119. A subexponential algorithm for Cluster Editing, combined with the reduction shown in Lemma 134, would give a subexponential (in the number of variables and clauses) algorithm for verifying satisfiability of 3-CNF formulas. Existence of such algorithm contradicts ETH by Corollary 5. □

We note that the graph constructed in the proof of Lemma 134 is of maximum degree 5. Thus our reduction intuitively shows that sparse instances of Cluster Editing where in the output the clusters are of constant size are already hard.

8.6 Conclusion and open questions

In this chapter we have given an algorithm that solves p-Cluster Editing in time $\mathcal{O}(2^{\mathcal{O}(\sqrt{pk})} + n + m)$ and complemented it by a multivariate lower bound, which shows that the running time of the presented algorithm is asymptotically tight for all p sublinear in k.

In our multivariate lower bound it is crucial that the cliques and clusters are arranged in groups of six. However, the drawback of this construction is that Theorem 116 settles the time complexity of p-Cluster Editing problem only for $p \geq 6$ (Corollary 118). It does not seem unreasonable that, for example, the 2-Cluster Editing problem, already NP-complete [299], may have enough

structure to allow an algorithm with running time $\mathcal{O}(2^{o(\sqrt{k})} + n + m)$. Can we construct such an algorithm or refute its existence under ETH?

Secondly, we would like to point out an interesting link between the subexponential parameterized complexity of the problem and its approximability. When the number of clusters drops from linear to sublinear in k, we obtain a phase transition in parameterized complexity from exponential to subexponential. As far as approximation is concerned, we know that bounding the number of clusters by a constant allows us to construct a PTAS [167], whereas the general problem is APX-hard [61]. The mutual drop of the parameterized complexity of a problem — from exponential to subexponential — and of approximability — from APX-hardness to admitting a PTAS — can be also observed for many hard problems when the input is constrained by additional topological bounds, for instance excluding a fixed pattern as a minor [104, 105, 140]. It is therefore an interesting question, whether p-Cluster Editing also admits a PTAS when one assumes that the target number of clusters is bounded by a non-constant, yet sublinear function of the optimum number of modifications k, for instance $p = \sqrt{k}$.

Chapter 9

Tight bounds for parameterized complexity of Edge Clique Cover

9.1 Introduction

As described in Chapter 4, recently there is much interest in pursuing tight bounds on parameterized complexity of classical NP-hard problems. The Exponential Time Hypothesis of Impagliazzo and Paturi [195] provides solid complexity foundations for proving sharp lower bounds, which in many cases essentially match known upper bounds. Such lower bounds often follow immediately from known NP-hardness reductions, when one considers classical problems that admit algorithms working in simple single exponential time in term of a natural parameter. However, more involved frameworks for different parameterizations and running times have been developed. Among others, reduction frameworks have been proposed for slightly superexponential running time [241], i.e., $\mathcal{O}^*(2^{\mathcal{O}(k \log k)})$, as well as for parameterizations by treewidth, for both simple single exponential and slightly superexponential running time [91, 239, 241]. A class of problems receiving a lot of attention recently are problems admitting *subexponential* parameterized algorithms, i.e., working in time $\mathcal{O}^*(2^{o(k)})$ for k being the parameter. We have already seen examples of such algorithms in Chapter 6 and in Chapter 8.

In this chapter we make one step further in the quest of finding tight runtime bounds for parameterized problems by presenting (to the best of our knowledge) the first natural double-exponential tight bound. Our problem of interest is an important combinatorial problem called Edge Clique Cover.

Edge Clique Cover	
Input:	An undirected graph G and a non-negative integer k.
Parameter:	k
Question:	Does there exist a set of k subgraphs of G, such that each subgraph is a clique and each edge of G is contained in at least one of these subgraphs?

Edge Clique Cover is known to be NP-complete even in very restricted graph classes [59, 191, 260] and was widely studied under a few different names: Covering by Cliques (GT17) [161], Intersection Graph Basis (GT59) [161] and Keyword Conflict [213]. It is known that the Edge Clique Cover problem is equivalent to the problem of finding a representation of a

graph G as an intersection model with at most k elements in the universe [123, 178, 278]. Therefore, a covering of a complex real-world network by a small number of cliques may reveal its hidden structure [180]. Further applications of Edge Clique Cover can be found in such various areas as computational geometry [4], applied statistics [170, 267], and compiler optimization [273]. Due to its importance, the Edge Clique Cover problem was studied from various perspectives, including approximation upper and lower bounds [19, 245], heuristics [26, 170, 213, 223, 267, 273] and polynomial-time algorithms for special graph classes [191, 192, 247, 260]. The references presented in this paragraph are a compilation of the literature review given by Gramm et al. [169] for the problem.

From the point of view of exact algorithms, a natural parameterization of Edge Clique Cover by the number of cliques was studied by Gramm et al. [169]. The authors propose a set of simple rules that reduce the number of vertices of the input graph to 2^k, that is, they provide a kernel with at most 2^k vertices. Currently the best known fixed-parameter algorithm for Edge Clique Cover parameterized by k is a straightforward dynamic programming on the 2^k-vertex kernel, which runs in double-exponential time in terms of k. Due to the importance of the Edge Clique Cover problem on one hand, and the lack of any improvement upon the very simple approach of Gramm et al. [169] on the other hand, Edge Clique Cover became a seasoned veteran of open problem sessions (with the most recent occurrence on the Workshop on Kernels in Vienna, 2011).

The first improvement came only recently, when together with Cygan, Kratsch, Pilipczuk, and Wahlström [89] we have shown that Edge Clique Cover is OR-compositional, refuting (under the assumption that NP $\not\subseteq$ coNP/poly) existence of a polynomial kernel for the problem. First, an AND-composition for the problem had been presented, which was then modified to an OR-composition. As AND-conjecture has been later proved to follow from assumption NP $\not\subseteq$ coNP/poly by Drucker [120], both an AND-composition and an OR-composition refutes existence of a polynomial kernel under NP $\not\subseteq$ coNP/poly. The results of Drucker were not known at the time of preparation of [89], so it was essential to show also an OR-composition in order to ground the result on an established conjecture that NP $\not\subseteq$ coNP/poly, rather than on AND-conjecture that was still speculated at this point.

Our results. In this chapter, based on paper [94], we complete the picture of the parameterized complexity of Edge Clique Cover. Since the results of the aforementioned previous work [89] are superseded by the results presented here, we choose to omit their description. The main technical result of this chapter is the following reduction.

Theorem 135. *There exists a polynomial-time algorithm that, given a 3-*CNF-SAT *formula with n variables and m clauses, constructs an equivalent* Edge Clique Cover *instance (G, k) with $k = \mathcal{O}(\log n)$ and $|V(G)| = \mathcal{O}(n + m)$.*

The above theorem immediately gives the following lower bound.

Corollary 136. *Unless ETH fails, there does not exist an algorithm solving* Edge Clique Cover *in time $\mathcal{O}^*(2^{2^{o(k)}})$.*

Proof. Assume that such an algorithm exists, and consider pipelining it with the reduction of Theorem 135. We obtain an algorithm for 3-CNF-SAT working in $2^{o(n)}$ time, which contradicts ETH. □

We note that ETH is not necessary to refute existence of single exponential algorithms for the problem; the same reduction proves also the following lower bound. Recall that a problem is solvable in *quasipolynomial time* if it admits an algorithm running in time $2^{\mathcal{O}(\log^{\mathcal{O}(1)} n)}$, where n is the total input length.

Corollary 137. *Unless all problems in NP are solvable in quasipolynomial time, there does not exist an algorithm solving* Edge Clique Cover *in time* $\mathcal{O}^*(2^{\mathcal{O}(k^{\mathcal{O}(1)})})$.

Proof. Assume that such an algorithm exists, and consider pipelining it with the reduction of Theorem 135. We obtain an algorithm for 3-CNF-SAT working in $2^{\mathcal{O}(\log^{\mathcal{O}(1)} n)}$ time, which means that all problems in NP are solvable in quasipolynomial time. □

We now show that our reduction can be used to prove that Edge Clique Cover does not admit a subexponential kernel even under the assumption that P≠NP. We remark that in [94] we claimed the following hardness result only under the assumption of ETH, and under the assumption of NP $\not\subseteq$ coNP/poly.

Corollary 138. *There exists a universal constant* $\varepsilon > 0$ *such that, unless P=NP, there is no constant* λ *and a polynomial-time algorithm* B *that takes an instance* (G, k) *of* Edge Clique Cover*, and outputs an equivalent instance* (G', k') *of* Edge Clique Cover *with binary encoding of length at most* $\lambda \cdot 2^{\varepsilon k}$.

Proof. Let us fix some binary encoding of instances of the 3-CNF-SAT problem such that an instance with n variables has encoding of length at least n. In the following, we also say *bitsize* for the length of the binary encoding. Aiming towards a contradiction, assume that such a constant λ and algorithm B exist for some small $\varepsilon > 0$, to be determined later. Let A be the algorithm given by Theorem 135; that is, A takes an instance Φ of 3-CNF-SAT on n variables, and returns an instance $A(\Phi)$ of Edge Clique Cover with parameter $k \leq \kappa_1 \cdot \log n$ for some universal constant κ_1. Moreover, since the unparameterized version of Edge Clique Cover belongs to NP, and 3-CNF-SAT is NP-complete, there exists a polynomial-time reduction from Edge Clique Cover to 3-CNF-SAT. More precisely, there exists an algorithm C that for a given instance (G, k) of Edge Clique Cover with bitsize at most m, returns an equivalent instance of 3-CNF-SAT with bitsize at most $\kappa_2 \cdot m^{\kappa_3}$, where κ_2, κ_3 are some universal constants.

Let now D be an algorithm obtained by pipelining algorithms A, B, C in this order. Clearly, since A, B, C work in polynomial-time, then so does D. Moreover, D takes as input an instance of 3-CNF-SAT, and outputs an equivalent instance of 3-CNF-SAT. We now examine how the size of the instance at hand changes when the algorithms A, B, C are applied consecutively in the algorithm D.

Assume that algorithm D is given an instance Φ with bitsize exactly n. Therefore, the number of variables in Φ is at most n, and the application of A turns Φ into an equivalent instance (G, k) of Edge Clique Cover, where $|G|$ is polynomial in n and $k \leq \kappa_1 \cdot \log n$. Next, we apply the kernelization algorithm B to (G, k) and obtain an equivalent instance (G', k') of Edge Clique Cover with bitsize at most $\lambda \cdot 2^{\varepsilon \cdot \kappa_1 \cdot \log n} = \lambda \cdot n^{\varepsilon \cdot \kappa_1}$. Finally, we apply the reduction C and obtain an equivalent instance Φ' of 3-CNF-SAT with bitsize at most $\kappa_2 \cdot (\lambda \cdot n^{\varepsilon \cdot \kappa_1})^{\kappa_3} = (\kappa_2 \cdot \lambda^{\kappa_3}) \cdot n^{\varepsilon \cdot \kappa_1 \cdot \kappa_3}$.

Let us now take any $\varepsilon > 0$ such that $\varepsilon < \frac{1}{\kappa_1 \cdot \kappa_3}$; recall that κ_1, κ_3 are universal constant, so ε is also a universal constant. Thus, the algorithm D takes an instance of 3-CNF-SAT with bitsize n, and returns an equivalent instance of 3-CNF-SAT with bitsize at most $(\kappa_2 \cdot \lambda^{\kappa_3}) \cdot n^{\varepsilon \cdot \kappa_1 \cdot \kappa_3}$. Assume that $n > \kappa_0$, where $\kappa_0 = (k_2 \cdot \lambda^{k_3})^{\frac{1}{1 - \varepsilon \cdot \kappa_1 \cdot \kappa_3}}$. Then,

$$n = n^{1 - \varepsilon \cdot \kappa_1 \cdot \kappa_3} \cdot n^{\varepsilon \cdot \kappa_1 \cdot \kappa_3} > (k_2 \cdot \lambda^{k_3}) \cdot n^{\varepsilon \cdot \kappa_1 \cdot \kappa_3}$$

Therefore, the algorithm D outputs an instance with strictly shorter binary encoding unless $n \leq (k_2 \cdot \lambda^{k_3})^{\frac{1}{1-\varepsilon\cdot\kappa_1\cdot\kappa_3}}$.

Consider now the following algorithm for the 3-CNF-SAT problem. Take an instance of the 3-CNF-SAT problem, and consecutively apply algorithm D to it until the bitsize of the instance at hand shrinks to at most κ_0. Then, apply any brute-force algorithm for the 3-CNF-SAT problem to resolve the instance in constant time. Note that in each application of algorithm D the bitsize of the instance at hand gets reduced by at least 1, so D is applied at most n times, where n is the bitsize of the original input instance. Each application of D runs in polynomial time, so the whole presented algorithm runs in polynomial time as well. Thus we have solved the 3-CNF-SAT problem in polynomial time, which contradicts the assumption that P≠NP. □

Finally, we show that under a stronger assumption of NP $\not\subseteq$ coNP/poly we can refute existence of not only kernelization, but also of compression into any language L with subexponential guarantee on the output size. For this we use the results of Dell and van Melkebeek [102] on lower bounds for compressibility of the 3-CNF-SAT problem (see Theorem 9).

Corollary 139. *There exists a universal constant $\varepsilon > 0$ such that, unless NP $\subseteq$ coNP/poly, there is no constant λ, language $L \subseteq \{0,1\}^*$, and a polynomial-time algorithm B that, given an* Edge Clique Cover *instance (G,k), outputs an equivalent instance $B(G,k)$ of L (that is, $B(G,k) \in L$ iff (G,k) is a YES-instance) with bitsize at most $\lambda \cdot 2^{\varepsilon k}$.*

Proof. Again, fix some binary encoding of the instances of the 3-CNF-SAT problem such that an instance with n variables has encoding of length at least n. Aiming towards a contradiction, assume that such a constant λ, language L and algorithm B exist for some small $\varepsilon > 0$, to be determined later. Let A be the algorithm given by Theorem 135; that is, A takes an instance Φ of 3-CNF-SAT on n variables, and returns an instance $A(\Phi)$ of Edge Clique Cover with parameter $k \leq \kappa_1 \cdot \log n$ for some universal constant κ_1.

Consider now algorithm D obtained by pipelining algorithms A and B. Clearly, D works in polynomial time, and given an instance Φ of 3-CNF-SAT it outputs an equivalent instance of L. We now examine how the size of the instance at hand changes when the algorithms A and B are applied consecutively in the algorithm D.

Assume that algorithm D is given an instance Φ with bitsize exactly n. Therefore, the number of variables in Φ is at most n, and the application of A turns Φ into an equivalent instance (G,k) of Edge Clique Cover, where $|G|$ is polynomial in n and $k \leq \kappa_1 \cdot \log n$. Next, we apply the compression algorithm B to (G,k) and obtain an equivalent instance of L with bitsize at most $\lambda \cdot 2^{\varepsilon\cdot\kappa_1\cdot\log n} = \lambda \cdot n^{\varepsilon\cdot\kappa_1}$. Observe now, that if we fix any ε such that $0 < \varepsilon < \frac{3}{\kappa_1}$, then the output instance is of size $\mathcal{O}(n^{3-\varepsilon'})$ for $\varepsilon' = 3 - \varepsilon \cdot \kappa_1 > 0$. By Theorem 9, existence of algorithm D is a contradiction with the assumption NP $\not\subseteq$ coNP/poly. □

To the best of our knowledge, the only known kernelization lower bounds that use the assumption P≠NP instead of stronger assumption NP $\not\subseteq$ coNP/poly, are the lower bounds for linear kernels for problems on planar graphs obtained by the technique of Chen et al. [65]. Very roughly speaking, in this framework one proves that if the parameterized dual of a problem admits a kernel of size at most αk for some constant α, where k is the dual parameter, then the problem itself cannot admit a kernel of size βk for $\beta < 1 + \frac{1}{\alpha-1}$. The reason is that if both these kernelization algorithms existed, then one could apply them alternately so that the size of the instance at hand would decrease at each step until it becomes bounded by some constant. Thus, by showing a linear kernel for the

parametric dual we can at the same time exclude existence of very efficient kernelization algorithms for the problem itself. See [65, 224] for examples.

The approach presented in Corollary 138 has some similarities with the framework of Chen et al. [65]: the contradiction with P≠NP is obtained by showing an efficient algorithm that can always strictly reduce the size of the instance at hand until it becomes a constant. However, the logical structure of the technique is very different. While the approach of Chen et al. [65] alternates between the problem and its dual and the technical difficulty is construction of the kernel for the dual, in our proof we alternate between 3-CNF-SAT and Edge Clique Cover and the crucial part is to prove an efficient way of translating these problems to one another. Of course, the presented approach is also substantially different from the hardness results obtained using the composition framework of Bodlaender et al. [45], which hold under the assumption that NP $\not\subseteq$ coNP/poly.

Throughout the chapter we investigate the graph we denote as H_ℓ, which is isomorphic to a clique on 2^ℓ vertices with a perfect matching removed, called the cocktail party graph. The core idea of the proof of Theorem 135 is the observation that a graph H_ℓ is a hard instance for the Edge Clique Cover problem, at least from the point of view of the currently known algorithms. Such a graph, while being immune to the reductions of Gramm et al. [169], can be quite easily covered with 2ℓ cliques, and there are multiple solutions of such size. Moreover, it is non-trivial to construct smaller clique covers for H_ℓ (but they exist).

In fact, the optimum size of a clique cover of cocktail party graphs with $2n$ vertices is proved to be $\min(k : n \leq \binom{k-1}{\lceil k/2 \rceil})$ for all $n > 1$ by Gregory and Pullman [172]. Moreover Chang et al. study cocktail party graphs in terms of rankwidth, which they prove to be unbounded in case of edge clique graphs of cocktail party graphs [58] (we refer to their work for appropriate definitions).

9.2 Double-exponential lower bound

This section is devoted to the proof of Theorem 135. In the following, for a bit-string $\mathbf{c}$ by $\overline{\mathbf{c}}$ we denote its bitwise negation.

Recall that graph H_ℓ is defined as a clique on 2^ℓ vertices with a perfect matching removed. In Section 9.2.1 we analyze in details graphs H_ℓ. It turns out that there is a large family of clique covers of size 2ℓ, where the clique cover consists of pairs of cliques whose vertex sets are complements of each other. We refer to such pairs as to *clique twins* and a clique cover consisting of clique twins is a *twin clique cover*. In particular, given any clique C of size $2^{\ell-1}$ in graph H_ℓ, we can construct a twin clique cover of H_ℓ that contains C. Note that we have $2^{2^{\ell-1}}$ such starting cliques C in H_ℓ: for each edge of the removed perfect matching, we choose exactly one endpoint into the clique. In our construction $\ell = \Theta(\log n)$ and this clique C encodes the assignment of the variables in the input 3-CNF-SAT formula.

Section 9.2.2 contains the details of our construction. We construct a graph G with edge set $E(G)$ partitioned into two sets E^{imp} and E^{free}. The first set contains the important edges, that is, the ones that are nontrivial to cover and play important role in the construction. The second set contains edges that are covered for free; in the end we add simplicial vertices (i.e., with neighborhood being a clique) to the graph G to cover E^{free}. Note that without loss of generality we can assume that a closed neighborhood of a non-isolated simplicial vertex is included in any optimal clique cover; however, we need to take care that we do not cover any edge of E^{imp} with such a clique and that we use only $\mathcal{O}(\log n)$ simplicial vertices (as each such vertex adds a clique to the solution).

While presenting the construction in Section 9.2.2, we give informal explanations of the role of each gadget. In Section 9.2.3 we show formally how to translate a satisfying assignment of the input formula into a clique cover of the constructed graph, whereas the reverse translation is provided in Section 9.2.4.

9.2.1 Cocktail party graph

In this section we analyze graphs H_ℓ known as cocktail party graphs; see Figure 9.1 for an illustration for small values of ℓ. Recall that for an integer $\ell \geq 1$ graph H_ℓ is defined as a complete graph on 2^ℓ vertices with a perfect matching removed. Note that a maximum clique in H_ℓ has $2^{\ell-1}$ vertices (i.e., half of all the vertices), and contains exactly one endpoint of each non-edge of H_ℓ. Moreover, if C is a maximum clique in H_ℓ, so is its complement $V(H_\ell) \setminus C$. This motivates the following definition.

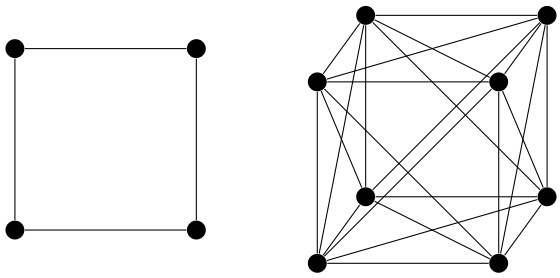

Figure 9.1: The graphs H_ℓ for $\ell = 2$ and $\ell = 3$. In the first case the optimum clique cover contains four two-vertex cliques and is a twin clique cover. In the second case an example twin clique cover is the set of all six faces of the cube; however, there exists a non-twin clique cover of H_3 of size five.

Definition 140. *A pair of maximum cliques C and $V(H_\ell) \setminus C$ in H_ℓ is called* clique twins. *A clique cover of H_ℓ that consists of clique twins is called a* twin clique cover.

The following lemma describes structure of twin clique covers of H_ℓ of size 2ℓ (i.e., containing ℓ twins).

Lemma 141. *Assume we are given a set $\mathcal{C}_0$ of $1 \leq \delta \leq \ell$ clique twins with the following property: if we choose one clique from each of the δ clique twins, the intersection of the vertex sets of these cliques has size exactly $2^{\ell-\delta}$. Then there exists a twin clique cover $\mathcal{C}$ of size 2ℓ that contains $\mathcal{C}_0$.*

Proof. We arbitrarily number the clique twins from $\mathcal{C}_0$ with numbers from 1 to δ, and in each clique twin we distinguish one clique labeled 0 and one labeled 1. Then, to each vertex $v \in V(H_\ell)$, we assign a δ-bit string $\mathbf{c}_v$ that on a position γ ($1 \leq \gamma \leq \delta$) contains the bit assigned to the clique that contains v from the γ-th clique twins. Since all subgraphs in $\mathcal{C}_0$ are cliques, for any non-edge uv of H_ℓ the strings $\mathbf{c}_u$ and $\mathbf{c}_v$ are bitwise negations.

Fix a δ-bit string $\mathbf{c}$ and consider $X_\mathbf{c} = \{v \in V(H_\ell) : \mathbf{c}_v = \mathbf{c}\}$. Note that any clique in $\mathcal{C}_0$ contains entire $X_\mathbf{c}$ or entire $X_{\overline{\mathbf{c}}}$ and, by the assumptions of the lemma, $|X_\mathbf{c}| = 2^{\ell-\delta}$. Moreover, as for a non-edge uv the strings $\mathbf{c}_u$ and $\mathbf{c}_v$ are bitwise negations, each non-edge of H_ℓ connects a vertex from $X_\mathbf{c}$, for some $\mathbf{c}$, with a vertex from $X_{\overline{\mathbf{c}}}$. We can now label each vertex $v \in X_\mathbf{c}$ with bit string $\mathbf{c}'_v$ of length $(\ell - \delta)$, such that in $X_\mathbf{c}$ each vertex receives a different label, and if uv is a non-edge of H_ℓ, then $\mathbf{c}'_u = \overline{\mathbf{c}'_v}$. In this manner each vertex $v \in V(H_\ell)$ receives a unique ℓ-bit label $\mathbf{c}_v\mathbf{c}'_v$ and for any non-edge uv of H_ℓ we have $\mathbf{c}_u\mathbf{c}'_u = \overline{\mathbf{c}_v\mathbf{c}'_v}$.

For an integer $1 \leq \gamma \leq \ell$ and a bit $c \in \{0,1\}$, consider a set $C_{\gamma,c}$ consisting of those vertices of H_ℓ whose aforementioned ℓ-bit labels have γ-th bit set to c. As in H_ℓ a vertex v is connected with all other vertices except the one labeled with the bitwise negation of the label of v, $C_{\gamma,c}$ induces a clique. Moreover, for any edge $uv \in E(H_\ell)$, the labels of u and v agree on at least one bit, and the corresponding clique $C_{\gamma,c}$ contains the edge uv. As $C_{\gamma,0} = V(H_\ell) \setminus C_{\gamma,1}$, we infer that the family $\mathcal{C} = \{C_{\gamma,c} : 1 \leq \gamma \leq \ell, c \in \{0,1\}\}$ is a twin clique cover of H_ℓ of size 2ℓ. We finish the proof of the lemma by noticing that $\{C_{\gamma,c} : 1 \leq \gamma \leq \delta, c \in \{0,1\}\} = \mathcal{C}_0$. □

Note that the above lemma for $\delta = 1$ implies that for any maximum clique C in H_ℓ there exists a twin clique cover of size 2ℓ that contains C. The next lemma treats about optimum twin clique covers of H_ℓ.

Lemma 142. *Let $\mathcal{C}$ be a clique cover of H_ℓ that contains at least $\ell - 1$ clique twins. Then $|\mathcal{C}| \geq 2\ell$ and, if $|\mathcal{C}| = 2\ell$, then $\mathcal{C}$ is a twin clique cover of H_ℓ.*

Proof. Let $\mathcal{C}_0 \subseteq \mathcal{C}$ be a set of $2\ell - 2$ cliques that form the assumed $\ell - 1$ clique twins. We use the family $\mathcal{C}_0$ to label the vertices of H_ℓ with $(\ell-1)$-bit labels as in the proof of Lemma 141. That is, we arbitrarily number these clique twins with numbers 1 to $\ell-1$, and in each clique twin we distinguish one clique labeled 0 and one labeled 1; a string $\mathbf{c}_v$ for $v \in V(H_\ell)$ consists of $(\ell-1)$ bits assigned to the cliques containing v. Again, for any non-edge uv of H_ℓ, the strings $\mathbf{c}_u$ and $\mathbf{c}_v$ are bitwise negations.

Fix a $(\ell-1)$-bit string $\mathbf{c}$ and consider $X_{\mathbf{c}} = \{v \in V(H_\ell) : \mathbf{c}_v = \mathbf{c}\}$. Note that any clique in $\mathcal{C}_0$ contains entire $X_{\mathbf{c}}$ or entire $X_{\overline{\mathbf{c}}}$ and thus no clique in $\mathcal{C}_0$ covers the edges of $E(X_{\mathbf{c}}, X_{\overline{\mathbf{c}}})$. Moreover, as for any non-edge uv we have $\mathbf{c}_u = \overline{\mathbf{c}_v}$, the sets $X_{\mathbf{c}}$ and $X_{\overline{\mathbf{c}}}$ are of equal size.

As $\mathcal{C}$ is a clique cover of H_ℓ, $\mathcal{C} \setminus \mathcal{C}_0$ covers $E(X_{\mathbf{c}}, X_{\overline{\mathbf{c}}})$. As $H_\ell[X_{\mathbf{c}} \cup X_{\overline{\mathbf{c}}}]$ is isomorphic to $K_{2|X_{\mathbf{c}}|}$ with a perfect matching removed, a direct check shows that if $|X_{\mathbf{c}}| \geq 3$ then $|\mathcal{C} \setminus \mathcal{C}_0| \geq 3$, that is, we need at least three cliques to cover $E(X_{\mathbf{c}}, X_{\overline{\mathbf{c}}})$. Thus, if $|\mathcal{C}| \leq 2\ell$ then for each string $\mathbf{c}$ we have $|X_{\mathbf{c}}| \leq 2$. As there are $2^{\ell-1}$ bit strings $\mathbf{c}$ and 2^ℓ vertices of H_ℓ, we infer that in this case $|X_{\mathbf{c}}| = 2$ for each bit string $\mathbf{c}$.

From now on assume that $|\mathcal{C}| \leq 2\ell$. Fix a bit string $\mathbf{c}$. Since $|X_{\mathbf{c}}| = 2$, then $H_\ell[X_{\mathbf{c}} \cup X_{\overline{\mathbf{c}}}]$ is isomorphic to a 4-cycle and $E(X_{\mathbf{c}}, X_{\overline{\mathbf{c}}})$ contains two opposite edges of this cycle. These edges cannot be covered with a single clique. We infer that $|\mathcal{C} \setminus \mathcal{C}_0| \geq 2$, i.e., $|\mathcal{C}| \geq 2\ell$. Hence $|\mathcal{C}| = 2\ell$ and let $\mathcal{C} \setminus \mathcal{C}_0 = \{C, C'\}$. Note that for any bit string $\mathbf{c}$ the clique C contains both endpoints of one edge of $E(X_{\mathbf{c}}, X_{\overline{\mathbf{c}}})$, and C' contains the endpoints of the second edge. Therefore $C = V(H_\ell) \setminus C'$ and the lemma follows. □

Let us remark that Lemma 142 implies that one cannot cover the graph H_ℓ with less than ℓ clique twins, i.e., the bound given by Lemma 141 is tight. Indeed, assume that there exists a twin clique cover of H_ℓ using $\ell' < \ell$ clique twins. If necessary, copy some of the clique twins in order to obtain a cover that uses exactly $\ell - 1$ twins. However, from Lemma 142 we infer that this cover needs to contain in fact more cliques, a contradiction.

9.2.2 Construction

Recall that, given a 3-CNF-SAT formula Φ, we are to construct an equivalent Edge Clique Cover instance with the target number of cliques bounded logarithmically in the number of variables of Φ. We start with an empty graph G, and we subsequently add new gadgets to G. Recall that the edge set of G is partitioned into E^{free} and E^{imp}; at the end of this section we show how to cover

the set E^{free} with a small number of cliques, each induced by a closed neighborhood of a simplicial vertex. We refer to Figure 9.2 for an illustration of the construction.

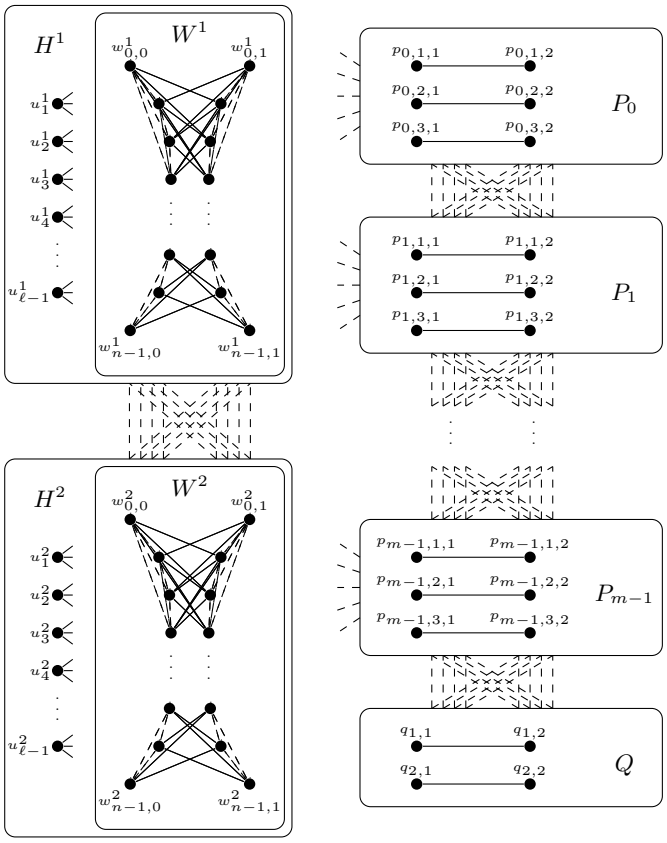

Figure 9.2: Illustration of the construction of graph G. Solid edges belong to the set E^{imp} and dashed edges to E^{free}. The simplicial vertices are not illustrated, nor are the details which edges between the clause gadgets P_j and the assignment gadgets H^η are present in the graph G. Moreover, not all edges of E^{free} between the vertices of gadgets Q and P_j are shown.

Preprocessing of the formula Φ

Let `Vars` denote the set of variables of Φ. By standard arguments, we can assume that in Φ each clause consists of exactly 3 literals, these literals contain different variables and no clause appears more than once. Moreover, we perform the following two regularization operations on Φ. First, we introduce some dummy variables into `Vars` so that the number of variables is a power of two, and that there exists at least one dummy variable (i.e., a variable that does not appear in any clause). This operation at most doubles the number of variables in `Vars`. Second, we ensure that if Φ is satisfiable, then there exists a satisfying assignment of Φ that assigns true to exactly half of the variables, and false to the other half. This can be done by transforming Φ into $\Phi' = \Phi \wedge \overline{\Phi}$, where $\overline{\Phi}$ is a copy of Φ on duplicated set of variables $\overline{\texttt{Vars}}$ and, moreover, all the literals in $\overline{\Phi}$ are replaced with their negations. Clearly, if Φ' has a satisfying assignment ϕ', then restricting ϕ' to `Vars` gives a satisfying assignment for Φ. Moreover, note that any assignment ϕ satisfying Φ can be extended

to an assignment ϕ' satisfying Φ' by assigning each copy of a variable the negation of the value of the original; the new assignment ϕ' assigns true to exactly half of the variables, and false to the other half. Observe that the second operation does not change the properties ensured by the first operation, and it exactly doubles the number of variables. Moreover, in the satisfying assignment we can fix value of one dummy variable in `Vars`.

After performing the described operations, let n be the number of variables in `Vars`, m be the number of clauses of Φ, and $n = 2^\ell$. Note that $m = \mathcal{O}(n^3)$ and $\log m = \mathcal{O}(\log n) = \mathcal{O}(\ell)$.

Assignment-encoding gadget

We assume that $\texttt{Vars} = \{x_0, x_1, \ldots, x_{n-1}\}$ and that the 0-th variable is a dummy one (it serves in the construction as a true pattern). Take a graph H isomorphic to $H_{\ell+1}$ and denote its vertices by $w_{i,c}$ for $0 \leq i < n$, $c \in \{0,1\}$; the non-edges of $H_{\ell+1}$ are $\{w_{i,0}w_{i,1} : 0 \leq i < n\}$. Let $W_c = \{w_{i,c} : 0 \leq i < n\}$ and $W = W_0 \cup W_1$. We put the edges of $H[W_0]$ and $H[W_1]$ into E^{free} and $E(W_0, W_1)$ into E^{imp}.

Moreover, we add $(\ell - 1)$ vertices u_γ, $1 \leq \gamma < \ell$, to the graph H. Each vertex u_γ is connected to all vertices of W via edges belonging to E^{imp}. This finishes the description of the assignment-encoding gadget H. We add two copies of the gadget H to the graph G, and denote the vertices of the η-th copy ($\eta \in \{1,2\}$) by $w^\eta_{i,c}$ and u^η_γ. We define W^η, W^η_0, W^η_1 and H^η in the natural way. In the graph G, for all indices $(i, c, i', c') \in (\{0, 1, \ldots, n-1\} \times \{0,1\})^2$ we connect each pair of vertices $w^1_{i,c}$ and $w^2_{i',c'}$ with an edge from E^{free}, i.e., we introduce a complete bipartite graph with edges belonging to E^{free} between W^1 and W^2.

Let us now describe the intuition behind this construction. In the gadget H, the neighborhood of each vertex u_γ is not a clique, thus any clique cover of H needs to include at least two cliques that contain u_γ. However, if we are allowed to use only two cliques per vertex u_γ, these cliques need to induce clique twins in the subgraph $H_{\ell+1}$ of H. With the assumption of only two cliques per vertex u_γ, the set of $(\ell - 1)$ vertices u_γ ensures that when covering $H_{\ell+1}$ we use ℓ clique twins: $(\ell - 1)$ from the vertices u_γ and one given by the edges in E^{free} (cliques W_0 and W_1). Lemma 142 asserts that the optimal way to complete a clique cover of $H_{\ell+1}$ is to use one more pair of clique twins: this clique twins, called *the assignment clique twins*, encode the assignment (and, as they are not bounded by the vertices u_γ, they can be used to verify the assignment in the forthcoming gadgets). Finally, we need two copies of the gadget H, as in the soundness proof we have one free clique that spoils the aforementioned budget-tightness argument; however, as the vertices $\{u^\eta_\gamma : 1 \leq \gamma < \ell, \eta \in \{1,2\}\}$ form an independent set, it cannot spoil it in both copies at the same time. The edges between the sets W in the copies allow us to use the same two cliques as the assignment clique twins in both copies of the gadget H.

One could ask why we put edges from $H[W_0]$ and $H[W_1]$ into E^{free}, since in the previous section we have assumed that all the edges of H_ℓ are to be covered. The reason for this is that additional cliques with several vertices from W_0 or W_1 will appear in order to cover other edges of E^{free}. Hence, we need to put edges from $H[W_0]$ and $H[W_1]$ into E^{free} to allow other cliques covering E^{free} to have larger intersections with W without covering any important edges.

Clause gadgets

We now introduce gadgets that verify correctness of the assignment encoded by the assignment clique twins, described in the previous paragraphs.

First, let us extend our notation. Let $\Phi = \Psi_0 \wedge \Psi_1 \wedge \ldots \wedge \Psi_{m-1}$ and for integers j, α let $i(j,\alpha)$ be the index of the variable that appears in α-th literal in the clause Ψ_j. Moreover, let $c(j,\alpha) = 0$ if the α-th literal of Ψ_j is negative (i.e., $\neg x_{i(j,\alpha)}$) and $c(j,\alpha) = 1$ otherwise.

For each clause Ψ_j, we introduce into G a subgraph P_j isomorphic to $3K_2$, that is, $V(P_j) = \{p_{j,\alpha,\beta} : 1 \leq \alpha \leq 3, \beta \in \{1,2\}\}$ and $E(P_j) = \{p_{j,\alpha,1}p_{j,\alpha,2} : 1 \leq \alpha \leq 3\}$. Moreover, we introduce into G a guard subgraph Q isomorphic to $2K_2$, that is, $V(Q) = \{q_{1,1}, q_{1,2}, q_{2,1}, q_{2,2}\}$ and $E(Q) = \{q_{1,1}q_{1,2}, q_{2,1}q_{2,2}\}$.

All the edges in all the gadgets P_j and Q belong to E^{imp}. Moreover, we introduce the following edges to E^{free}. First, for each vertex of Q, we connect it with all vertices of all the subgraphs P_j. Second, we connect each vertex $p_{j,\alpha,\beta}$ with all the vertices $p_{j',\alpha',\beta'}$ for $j' \neq j, 1 \leq \alpha' \leq 3, \beta' \in \{1,2\}$. Third, we connect each vertex $p_{j,\alpha,\beta}$ with all the vertices in the sets W in both copies of the gadget H, except for $w^{\eta}_{0,1}$ and $w^{\eta}_{i(j,\alpha),1-c(j,\alpha)}$ for $\eta \in \{1,2\}$. This finishes the description of the clause gadgets.

Let us now describe the intuition behind this construction. In each gadget P_j and in the guard subgraph Q the edges are independent, thus they need to be covered by different cliques. Two cliques are used to cover the edges of Q, and they can cover two out of three edges from each of the clause gadget P_j. The third one needs to be covered by the assignment clique twins from the gadgets H (as the gadgets P_j are not adjacent to the vertices u^{η}_{γ}), and it corresponds to the choice which literal satisfies the clause Ψ_j. The missing edges $p_{j,\alpha,\beta}w^{\eta}_{0,1}$ ensures that only one clique of the assignment clique twins is used to cover the edges of P_j. Finally, the missing edge $p_{j,\alpha,\beta}w^{\eta}_{i(j,\alpha),1-c(j,\alpha)}$ verifies that this clique encodes a satisfying assignment of Φ.

We note that it is non-trivial to cover the edges of E^{free} with $\mathcal{O}(\ell) = \mathcal{O}(\log n)$ cliques induced by closed neighborhoods of simplicial vertices. This is done in the next sections by appropriately using bit-vectors.

Budget

We set the number of cliques to cover the edges of E^{imp} as

$$k_0 = 2 \cdot 2 \cdot (\ell - 1) + 2 + 2 = 4\ell,$$

that is, two for each vertex u^{η}_{γ}, two for the assignment clique twins in H, and two for the cliques that contain the edges of Q. The final number of cliques k is the sum of k_0 and the number of simplicial vertices introduced in the next section.

Covering the free edges

In this section we show that the edges of E^{free} can be covered by a small number of cliques without accidentally covering any edge of E^{imp}. Moreover, such covering can be constructed in polynomial time. To construct the final Edge Clique Cover instance, for each clique of this clique cover we introduce a simplicial vertex adjacent to the vertex set of this clique, and raise the desired number of cliques by one.

Lemma 143. *The graph $G^{free} = (V(G), E^{free})$ admits a clique cover of size $46 + 36\lceil \log m \rceil + 24\ell = \mathcal{O}(\log n)$. Moreover, such a clique cover can be constructed in polynomial time.*

Proof. We start by noting that the free edges in the copies of the gadget H, and between these copies, can be covered by four cliques: $W^1_c \cup W^2_{c'}$ for $(c, c') \in \{0,1\} \times \{0,1\}$. Similarly, the edges

incident to the guard gadget Q can be covered with 24 cliques: $\{q_{\alpha,\beta}\} \cup \{p_{j,\alpha',\beta'} : 0 \leq j < m\}$ for $(\alpha, \beta, \alpha', \beta') \in \{1,2\} \times \{1,2\} \times \{1,2,3\} \times \{1,2\}$.

Covering the edges between different gadgets P_j requires a bit more work. For each $1 \leq \gamma \leq \lceil \log m \rceil$ and $(\alpha, \beta, \alpha', \beta') \in (\{1,2,3\} \times \{1,2\})^2$ we take a clique $C^P_{\gamma,\alpha,\beta,\alpha',\beta'}$ that contains exactly one vertex from each gadget P_j: if the γ-th bit of the binary representation of j equals 0, $p_{j,\alpha,\beta} \in C^P_{\gamma,\alpha,\beta,\alpha',\beta'}$, and otherwise $p_{j,\alpha',\beta'} \in C^P_{\gamma,\alpha,\beta,\alpha',\beta'}$. Clearly, $C^P_{\gamma,\alpha,\beta,\alpha',\beta'}$ induces a clique in G^{free}, as it contains exactly one vertex from each gadget P_j. Let us now verify that all edges between the gadgets P_j are covered by these $36\lceil \log m \rceil$ cliques. Take any edge $p_{j,\alpha,\beta}p_{j',\alpha',\beta'} \in E^{\text{free}}$, $j \neq j'$. Assume that the binary representations of j and j' differ on the γ-th bit; without loss of generality, assume that the γ-th bit of j is 0, and the γ-th bit of j' is 1. Then the clique $C^P_{\gamma,\alpha,\beta,\alpha',\beta'}$ contains both $p_{j,\alpha,\beta}$ and $p_{j',\alpha',\beta'}$.

We now handle the edges that connect the two copies of the gadget H with the gadgets P_j. First, we take care of the edges that are incident to the vertices $w^\eta_{0,0}$. This can be easily done with 6 cliques: for each $(\alpha, \beta) \in \{1,2,3\} \times \{1,2\}$ we take a clique that contains $w^1_{0,0}$, $w^2_{0,0}$ as well as all vertices $p_{j,\alpha,\beta}$ for $0 \leq j < m$. Second, we take care of the edges $p_{j,\alpha,\beta}w^\eta_{i(j,\alpha),c(j,\alpha)}$. To this end, we take 12 cliques: for each $(\alpha, \beta, c) \in \{1,2,3\} \times \{1,2\} \times \{0,1\}$ we take a clique that contains $w^\eta_{i,c}$ for $1 \leq i < n$, $\eta \in \{1,2\}$ as well as all the vertices $p_{j,\alpha,\beta}$ that satisfy $c(j,\alpha) = c$.

We are left with the edges of form $p_{j,\alpha,\beta}w^\eta_{i,c}$ for $i \notin \{0, i(j,\alpha)\}$. These edges can be covered in a similar fashion to the edges between the gadgets P_j. For each $1 \leq \gamma \leq \ell$ and $(\alpha, \beta, c, c') \in \{1,2,3\} \times \{1,2\} \times \{0,1\}^2$ we construct a clique $C^W_{\gamma,\alpha,\beta,c,c'}$ that contains all vertices $w^\eta_{i,c}$ for $\eta \in \{1,2\}$ and $1 \leq i < n$ such that the γ-th bit of the binary representation of i equals c', as well as all vertices $p_{j,\alpha,\beta}$ for $0 \leq j < m$ such that the γ-th bit of the binary representation of $i(j,\alpha)$ equals $1 - c'$. To see that $C^W_{\gamma,\alpha,\beta,c,c'}$ is indeed a clique in G^{free}, note that it contains only edges in $G[W^1_0 \cup W^2_0]$ or $G[W^1_1 \cup W^2_1]$, between different gadgets P_j, and edges of the form $p_{j,\alpha,\beta}w^\eta_{i,c}$ where $i \neq 0$ and $i \neq i(j,\alpha)$ (the indices i and $i(j,\alpha)$ must differ on the γ-th bit if both $p_{j,\alpha,\beta}$ and $w^\eta_{i,c}$ are included in the clique $C^W_{\gamma,\alpha,\beta,c,c'}$). We finish the proof of the lemma by verifying that all the edges of the form $p_{j,\alpha,\beta}w^\eta_{i,c}$ for $i \notin \{0, i(j,\alpha)\}$ are covered by these 24ℓ cliques. As $i \neq i(j,\alpha)$, there exists $1 \leq \gamma \leq \ell$ such that i and $i(j,\alpha)$ differ on the γ-th bit of their binary representations. Let c' be the γ-th bit of the binary representation of i. We infer that both $p_{j,\alpha,\beta}$ and $w^\eta_{i,c}$ are included in the clique $C_{\gamma,\alpha,\beta,c,c'}$ and the lemma is proven. □

Recall that for each clique constructed by Lemma 143 we add a simplicial vertex to G that is adjacent to all vertices of this clique. The simplicial vertices are independent in G. As discussed earlier, we can assume that for any non-isolated simplicial vertex s in G, any optimal clique cover in G contains a clique whose vertex set equals to the closed neighborhood of s.

We conclude the construction section by setting the desired number of cliques k to be the sum of k_0 and the number of aforementioned simplicial vertices, $k = 4\ell + 46 + 36\lceil \log m \rceil + 24\ell = \mathcal{O}(\log n)$.

9.2.3 Completeness

In this section we show how to translate a satisfying assignment of the input formula Φ into a clique cover of G of size k.

Lemma 144. *If the input formula Φ is satisfiable, then there exists a clique cover of the graph G of size k.*

Proof. Let $\phi : \{0, 1, \ldots, n-1\} \to \{0, 1\}$ be a satisfying assignment of Φ, that is, $\phi(i)$ is the value of x_i, 0 stands for false and 1 stands for true. By the properties of the preprocessing step of the construction, we may assume that $|\phi^{-1}(0)| = |\phi^{-1}(1)| = |\texttt{Vars}|/2 = 2^{\ell-1}$ and that $\phi(0) = 0$ (as x_0 is a dummy variable).

We start the construction of the clique cover $\mathcal{C}$ of the graph G by taking into $\mathcal{C}$, for each of the $46 + 36\lceil \log m \rceil + 24\ell$ simplicial vertices of G constructed in Lemma 143, a clique induced by the closed neighborhood of the simplicial vertex. In this manner we cover all the edges of E^{free}, and we are left with a budget of 4ℓ cliques.

We define the assignment clique twins C_0^A and C_1^A. For each clause Ψ_j of Φ, let $\alpha(j)$ be an index of a literal that is satisfied by ϕ in Ψ_j (if there is more than one such literal, we choose an arbitrary one). The clique C_0^A contains the vertices $w^\eta_{i,\phi(i)}$ for $0 \leq i < n$ and $\eta \in \{1, 2\}$ as well as the following vertices from the clause gadgets: $p_{j,\alpha(j),\beta}$ for $0 \leq j < m$, $\beta \in \{1, 2\}$. Note that C_0^A is indeed a clique, since the only missing edges between vertices $w^\eta_{i,c}$ and $p_{j,\alpha,\beta}$ are either incident to $w^\eta_{0,1}$ (but $\phi(0) = 0$) or of the form $w^\eta_{i(j,\alpha),1-c(j,\alpha)}p_{j,\alpha,\beta}$ (but $\phi(i(j, \alpha(j)))$ satisfies the $\alpha(j)$-th literal of Ψ_j, i.e., $c(j, \alpha(j)) = \phi(i(j, \alpha(j)))$).

The clique C_1^A is the twin (complement) of the clique C_0^A in both copies of the graph $H_{\ell+1}$, i.e., $C_1^A = \{w^\eta_{i,1-\phi(i)} : 0 \leq i < n, \eta \in \{1, 2\}\}$. Clearly, C_1^A is a clique in G.

Let us now fix $\eta \in \{1, 2\}$ and focus on the graph $G[W^\eta]$ isomorphic to $H_{\ell+1}$. The edges of E^{free} in this subgraph form clique twins $\{w^\eta_{i,c} : 0 \leq i < n\}$ for $c \in \{0, 1\}$. The assignment clique twins C_0^A and C_1^A form second clique twins in $G[W^\eta]$, after truncating them to this subgraph. Moreover, the assumption that ϕ evaluates exactly half of the variables to false and half to true implies that these two clique twins satisfy the assumptions of Lemma 141. We infer that all remaining edges of $G[W^\eta]$ can be covered by $\ell - 1$ clique twins; we add the vertex u^η_γ to both cliques of the γ-th clique twin, and add these clique twins to the constructed clique cover $\mathcal{C}$. In this manner we cover all the edges incident to all the vertices u^η_γ for $1 \leq \gamma < \ell$, $\eta \in \{1, 2\}$. As we perform this construction for both values $\eta \in \{1, 2\}$, we use $4\ell - 4$ cliques, and we are left with a budget of two cliques.

The cliques introduced in the previous paragraph cover all the edges in E^{imp} in both copies of the assignment gadget H. We are left with the clause gadgets P_j and the guard gadget Q. Recall that the clique C_0^A covers one out of three edges in each gadget P_j. Hence, it is straightforward to cover the remaining edges with two cliques: each clique contains both endpoints of exactly one uncovered edge from each gadget P_j and from the gadget Q. This finishes the proof of the lemma. □

9.2.4 Soundness

In this section we show a reverse transformation: a clique cover of G of size at most k cannot differ much from the one constructed in the proof of Lemma 144 and, therefore, encodes a satisfying assignment of the input formula Φ.

Lemma 145. *If there exists a clique cover of G of size at most k, then the input formula Φ is satisfiable.*

Proof. Let $\mathcal{C}$ be a clique cover of size at most k of G. As G contains $k - 4\ell$ simplicial vertices, without loss of generality we may assume that, for each such simplicial vertex s, the family $\mathcal{C}$ contains a clique induced by the closed neighborhood of s. These cliques cover the edges of E^{free}, but no edge of E^{imp}. Let $\mathcal{C}_0 \subseteq \mathcal{C}$ be the set of the remaining cliques; $|\mathcal{C}_0| \leq 4\ell$.

Let us start with analyzing the guard gadget Q. It contains two independent edges from E^{imp}. Thus, $\mathcal{C}_0$ contains two cliques, each containing one edge of Q. Denote this cliques by C_1^Q and C_2^Q.

Note that each clause gadget P_j contains three independent edges, and only two of them may be covered by the cliques C_1^Q and C_2^Q. Thus there exists at least one additional clique in $\mathcal{C}_0$ that contains an edge of at least one gadget P_j; let us denote this clique by C_0^A (if there is more than one such clique, we choose an arbitrary one).

Each vertex u_γ^η for $1 \leq \gamma < \ell$, $\eta \in \{1,2\}$, needs to be contained in at least two cliques of $\mathcal{C}_0$, since all its incident edges are in E^{imp} and the neighborhood of u_γ^η is not a clique. Moreover, no vertex u_γ^η may belong to C_1^Q, C_2^Q nor to C_0^A, as these vertices are not adjacent to the vertices of P_j and Q. As there are $2\ell - 2$ vertices u_γ^η, the vertices u_γ^η are independent, and $|\mathcal{C}_0 \setminus \{C_1^Q, C_2^Q, C_0^A\}| \leq 4\ell - 3$, we infer that at most one vertex u_γ^η may be contained in more than two cliques from $\mathcal{C}_0$. Without loss of generality we can assume that this vertex belongs to the second copy of the assignment gadget H, that is, each vertex u_γ^1 for $1 \leq \gamma < \ell$ belongs to exactly two cliques $C_{\gamma,0}^U, C_{\gamma,1}^U \in \mathcal{C}_0$.

Note that the only way to cover the edges incident to u_γ^1 with only two cliques $C_{\gamma,0}^U$, $C_{\gamma,1}^U$ is to take these cliques to induce clique twins in $G[W^1] \cong H_{\ell+1}$. That is, $C_{\gamma,0}^U$ consists of u_γ^1 and exactly one endpoint of each non-edge of $G[W^1]$, and $C_{\gamma,1}^U = \{u_\gamma^1\} \cup (W^1 \setminus C_{\gamma,0}^U)$.

We infer that the cliques $\{C_{\gamma,c}^U : 1 \leq \gamma < \ell, c \in \{0,1\}\}$, together with the clique twins formed by the edges of E^{free} in $G[W^1]$, induce ℓ clique twins in $G[W^1] \cong H_{\ell+1}$. Moreover, the remaining edges of $G[W^1]$ need to be covered with only two additional cliques (including C_0^A): there are at least $4\ell - 4$ cliques that contain vertices u_γ^1 or u_γ^2, out of which exactly $2l - 2$ can contain vertices from W^1, while at least two other cliques have to contain vertices of Q thus having to be disjoint with W^1. Lemma 142 asserts that the only way to cover the edges of $G[W^1] \cong H_{\ell+1}$ is to use one additional pair of cliques that are clique twins; thus, $C_0^A \cap W^1$ is a maximum clique in $G[W^1]$ and $\mathcal{C}_0$ contains also a clique C_1^A such that $C_1^A \cap W^1 = W^1 \setminus C_0^A$.

Recall that the clique C_0^A contained an edge from at least one gadget P_j. Therefore, $x_{0,1}^1 \notin C_0^A$, as $x_{0,1}^1$ is not adjacent to any vertex in any gadget P_j. Since C_0^A and C_1^A induce clique twins in $G[W_1]$, we infer that $x_{0,1}^1 \in C_1^A$ and C_1^A is disjoint with all the gadgets P_j. As the vertices u_γ^η are not adjacent to the gadgets P_j, the cliques that cover the edges incident to the vertices u_γ^η cannot cover any edges of the gadgets P_j either. We infer that the edges of the gadgets P_j are covered by only three cliques — C_0^A, C_1^Q and C_2^Q — and that in each gadget P_j, each of this three cliques contains exactly one edge. For a clause Ψ_j, let $\alpha(j)$ be the index of the literal whose edge is covered by C_0^A.

We claim that an assignment $\phi : \{0, 1, \ldots, n-1\} \to \{0,1\}$ that assigns the value $\phi(i)$ to the variable x_i in such a manner that $w_{i,\phi(i)}^1 \in C_0^A$, satisfies the formula Φ. More precisely, we claim that for each clause Ψ_j, the $\alpha(j)$-th literal of Ψ_j satisfies this clause in the assignment ϕ. Indeed, as $p_{j,\alpha(j),\beta} \in C_0^A$ for $\beta \in \{1,2\}$, and $p_{j,\alpha(j),\beta}$ is not adjacent to $w_{i(j,\alpha(j)),1-c(j,\alpha(j))}^1$, we have that $w_{i(j,\alpha(j)),c(j,\alpha(j))}^1 \in C_0^A$ and $\phi(i(j,\alpha(j))) = c(j,\alpha(j))$. This finishes the proof of the lemma and concludes the proof of Theorem 135. □

Bibliography

[1] Isolde Adler. Directed tree-width examples. *J. Comb. Theory, Ser. B*, 97(5):718–725, 2007.

[2] Isolde Adler, Stavros G. Kolliopoulos, Philipp Klaus Krause, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Tight bounds for linkages in planar graphs. In *ICALP 2011 (1)*, volume 6755 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 2011.

[3] Donald L. Adolphson and T. C. Hu. Optimal linear ordering. *SIAM J. Appl. Math.*, 25(3):403–423, 1973.

[4] Pankaj K. Agarwal, Noga Alon, Boris Aronov, and Subhash Suri. Can visibility graphs be represented compactly? *Discrete & Computational Geometry*, 12:347–365, 1994.

[5] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5), 2008.

[6] Jochen Alber, Hans L. Bodlaender, Henning Fernau, Ton Kloks, and Rolf Niedermeier. Fixed parameter algorithms for Dominating Set and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002.

[7] Jochen Alber, Henning Fernau, and Rolf Niedermeier. Parameterized complexity: exponential speed-up for planar graph problems. *J. Algorithms*, 52(1):26–56, 2004.

[8] Jochen Alber and Jiří Fiala. Geometric separation and exact solutions for the parameterized independent set problem on disk graphs. *J. Algorithms*, 52(2):134–151, 2004.

[9] Noga Alon. Ranking tournaments. *SIAM J. Discrete Math.*, 20(1):137–142, 2006.

[10] Noga Alon, Daniel Lokshtanov, and Saket Saurabh. Fast FAST. In *ICALP (1)*, volume 5555 of *Lecture Notes in Computer Science*, pages 49–58. Springer, 2009.

[11] Noga Alon, Konstantin Makarychev, Yury Makarychev, and Assaf Naor. Quadratic forms on graphs. In *STOC 2005*, pages 486–493. ACM, 2005.

[12] Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms*, 2(2):153–177, 2006.

[13] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[14] Archiwum olimpiady matematycznej. http://archom.ptm.org.pl/, in Polish.

[15] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.

[16] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

[17] Sanjeev Arora, Eli Berger, Elad Hazan, Guy Kindler, and Muli Safra. On non-approximability for quadratic programs. In *FOCS 2005*, pages 206–215. IEEE Computer Society, 2005.

[18] Bengt Aspvall, Michael F. Plass, and Robert E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979.

[19] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.

[20] Per Austrin, Toniann Pitassi, and Yu Wu. Inapproximability of treewidth, one-shot pebbling, and related layout problems. In *APPROX-RANDOM 2012*, volume 7408 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2012.

[21] Jørgen Bang-Jensen. Edge-disjoint in- and out-branchings in tournaments and related path problems. *J. Comb. Theory, Ser. B*, 51(1):1–23, 1991.

[22] Jørgen Bang-Jensen and Gregory Gutin. *Digraphs - theory, algorithms and applications*. Springer Monographs in Mathematics. Springer-Verlag London Ltd., London, second edition, 2009.

[23] Jørgen Bang-Jensen and Carsten Thomassen. A polynomial algorithm for the 2-path problem for semicomplete digraphs. *SIAM J. Discrete Math.*, 5(3):366–376, 1992.

[24] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56:89–113, 2004.

[25] János Barát. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22(2):161–172, 2006.

[26] Michael Behrisch and Anusch Taraz. Efficiently covering complex networks with cliques of similar vertices. *Theoretical Computer Science*, 355(1):37–47, 2006.

[27] Rémy Belmonte, Pim van 't Hof, and Marcin Kamiński. Induced immersions. In *ISAAC 2012*, volume 7676 of *Lecture Notes in Computer Science*, pages 299–308. Springer, 2012.

[28] Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4):281–297, 1999.

[29] Dietmar Berwanger, Anuj Dawar, Paul Hunter, and Stephan Kreutzer. DAG-width and parity games. In *STACS 2006*, volume 3884 of *Lecture Notes in Computer Science*, pages 524–536. Springer, 2006.

[30] Dietmar Berwanger, Anuj Dawar, Paul Hunter, Stephan Kreutzer, and Jan Obdržálek. The DAG-width of directed graphs. *J. Comb. Theory, Ser. B*, 102(4):900–923, 2012.

[31] Dietmar Berwanger and Erich Grädel. Entanglement - a measure for the complexity of directed graphs with applications to logic and games. In *LPAR 2004*, volume 3452 of *Lecture Notes in Computer Science*, pages 209–223. Springer, 2004.

[32] Stéphane Bessy, Fedor V. Fomin, Serge Gaspers, Christophe Paul, Anthony Perez, Saket Saurabh, and Stéphan Thomassé. Kernels for Feedback Arc Set in Tournaments. *J. Comput. Syst. Sci.*, 77(6):1071–1078, 2011.

[33] Andreas Björklund. Determinant sums for undirected hamiltonicity. In *FOCS 2010*, pages 173–182. IEEE Computer Society, 2010.

[34] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: Fast subset convolution. In *STOC 2007*, pages 67–74. ACM, 2007.

[35] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Trimmed Moebius inversion and graphs of bounded degree. *Theory Comput. Syst.*, 47(3):637–654, 2010.

[36] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009.

[37] Ivan Bliznets, Fedor V. Fomin, Michał Pilipczuk, and Yngve Villanger. Finding maximum induced chordal and interval graphs faster than 2^n. To appear in the proceedings of ESA 2013.

[38] Sebastian Böcker. A golden ratio parameterized algorithm for cluster editing. *Journal of Discrete Algorithms*, 16:79–89, 2012.

[39] Sebastian Böcker, Sebastian Briesemeister, Quang Bao Anh Bui, and Anke Truß. A fixed-parameter approach for weighted cluster editing. In *APBC 2008*, volume 6 of *Advances in Bioinformatics and Computational Biology*, pages 211–220, 2008.

[40] Sebastian Böcker, Sebastian Briesemeister, and Gunnar W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011.

[41] Sebastian Böcker and Peter Damaschke. Even faster parameterized cluster deletion and cluster editing. *Inf. Process. Lett.*, 111(14):717–721, 2011.

[42] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[43] Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. In *ICALP 2013 (1)*, volume 7965 of *Lecture Notes in Computer Science*, pages 196–207. Springer, 2013.

[44] 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.

[45] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

[46] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michał Pilipczuk. An $O(c^k \cdot n)$ 5-approximation algorithm for treewidth. *CoRR*, abs/1304.6321, 2013. To appear in the proceedings of FOCS 2013.

[47] Hans L. Bodlaender, Michael R. Fellows, Pinar Heggernes, Federico Mancini, Charis Papadopoulos, and Frances A. Rosamond. Clustering with partial information. *Theoretical Computer Science*, 411(7-9):1202–1211, 2010.

[48] Hans L. Bodlaender, Michael R. Fellows, and Dimitrios M. Thilikos. Derivation of algorithms for cutwidth and related graph layout parameters. *J. Comput. Syst. Sci.*, 75(4):231–244, 2009.

[49] Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) kernelization. In *FOCS 2009*, pages 629–638. IEEE Computer Society, 2009.

[50] Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Cross-composition: A new technique for kernelization lower bounds. In *STACS 2011*, volume 9 of *LIPIcs*, pages 165–176. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[51] Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

[52] Hans L. Bodlaender, Erik Jan van Leeuwen, Johan M. M. van Rooij, and Martin Vatshelle. Faster algorithms on branch and clique decompositions. In *MFCS 2010*, volume 6281 of *Lecture Notes in Computer Science*, pages 174–185. Springer, 2010.

[53] 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(5&6):555–581, 1992.

[54] Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. *SIAM J. Comput.*, 22(3):560–572, 1993.

[55] Liming Cai and David W. Juedes. On the existence of subexponential parameterized algorithms. *J. Comput. Syst. Sci.*, 67(4):789–807, 2003.

[56] Chris Calabro, Russell Impagliazzo, Valentine Kabanets, and Ramamohan Paturi. The complexity of Unique k-SAT: An isolation lemma for k-CNFs. *J. Comput. Syst. Sci.*, 74(3):386–393, 2008.

[57] Yixin Cao and Jianer Chen. Cluster editing: Kernelization based on edge cuts. *Algorithmica*, 64(1):152–169, 2012.

[58] Maw-Shang Chang, Ton Kloks, and Ching-Hao Liu. Edge-clique graphs of cocktail parties have unbounded rankwidth. *CoRR*, abs/1205.2483, 2012.

[59] Maw-Shang Chang and Haiko Müller. On the tree-degree of graphs. In *WG 2001*, pages 44–54, 2001.

[60] Pierre Charbit, Stéphan Thomassé, and Anders Yeo. The minimum feedback arc set problem is np-hard for tournaments. *Combinatorics, Probability & Computing*, 16(1):1–4, 2007.

[61] Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005.

[62] Moses Charikar and Anthony Wirth. Maximizing quadratic programs: Extending Grothendieck's inequality. In *FOCS 2004*, pages 54–60. IEEE Computer Society, 2004.

[63] Chandra Chekuri and Julia Chuzhoy. Polynomial bounds for the grid-minor theorem. *CoRR*, abs/1305.6577, 2013.

[64] Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David W. Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation*, 201(2):216–231, 2005.

[65] Jianer Chen, Henning Fernau, Iyad A. Kanj, and Ge Xia. Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. *SIAM J. Comput.*, 37(4):1077–1106, 2007.

[66] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. On the computational hardness based on linear FPT-reductions. *J. Comb. Optim.*, 11(2):231–247, 2006.

[67] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *J. Comput. Syst. Sci.*, 72(8):1346–1367, 2006.

[68] Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.

[69] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010.

[70] Jianer Chen and Jie Meng. A $2k$ kernel for the Cluster Editing problem. *J. Comput. Syst. Sci.*, 78(1):211–220, 2012.

[71] Phyllis Z. Chinn, Jarmila Chvátalová, Alexander K. Dewdney, and Norman E. Gibbs. The bandwidth problem for graphs and matrices — a survey. *J. Graph Theory*, 6:223–254, 1982.

[72] Maria Chudnovsky, Alexandra Fradkin, and Paul D. Seymour. Tournament immersion and cutwidth. *J. Comb. Theory Ser. B*, 102(1):93–101, 2012.

[73] Maria Chudnovsky, Alex Scott, and Paul D. Seymour. Vertex disjoint paths in tournaments, 2011. Manuscript.

[74] Maria Chudnovsky and Paul D. Seymour. A well-quasi-order for tournaments. *J. Comb. Theory, Ser. B*, 101(1):47–53, 2011.

[75] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC 1971*, pages 151–158. ACM, 1971.

[76] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[77] Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005.

[78] Bruno Courcelle. The Monadic Second-Order logic of graphs: Definable sets of finite graphs. In *WG 1988*, volume 344 of *Lecture Notes in Computer Science*, pages 30–53. Springer, 1988.

[79] Bruno Courcelle. The Monadic Second-Order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[80] Bruno Courcelle. The expression of graph properties and graph transformations in Monadic Second-Order logic. In *Handbook of Graph Grammars*, pages 313–400. World Scientific, 1997.

[81] Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci.*, 46(2):218–270, 1993.

[82] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

[83] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

[84] Bruno Courcelle and Sang-il Oum. Vertex-minors, Monadic Second-Order logic, and a conjecture by Seese. *J. Comb. Theory, Ser. B*, 97(1):91–126, 2007.

[85] Nigel J. Cutland. *Computability*. Cambridge University Press, 1980.

[86] Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. In *IEEE Conference on Computational Complexity 2012*, pages 74–84, 2012.

[87] Marek Cygan, Fabrizio Grandoni, and Danny Hermelin. Tight kernel bounds for problems on graphs with small degeneracy. *CoRR*, abs/1305.4914, 2013. To appear in the proceedings of ESA 2013.

[88] Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. In *STOC 2013*, pages 301–310. ACM, 2013.

[89] Marek Cygan, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Magnus Wahlström. Clique cover and graph separation: New incompressibility results. In *ICALP 2012 (1)*, pages 254–265, 2012.

[90] Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michał Pilipczuk. The planar directed k-vertex-disjoint paths problem is fixed-parameter tractable. *CoRR*, abs/1304.4207, 2013. To appear in the proceedings of FOCS 2013.

[91] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS 2011*, pages 150–159, 2011.

[92] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *CoRR*, abs/1103.0534, 2011.

[93] Marek Cygan and Marcin Pilipczuk. Faster exponential-time algorithms in graphs of bounded average degree. In *ICALP 2013 (1)*, volume 7965 of *Lecture Notes in Computer Science*, pages 364–375. Springer, 2013.

[94] Marek Cygan, Marcin Pilipczuk, and Michał Pilipczuk. Known algorithms for Edge Clique Cover are probably optimal. In *SODA 2013*, pages 1044–1053. SIAM, 2013.

[95] Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. Scheduling partially ordered jobs faster than 2^n. In *ESA 2011*, volume 6942 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2011.

[96] Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. Kernelization hardness of connectivity problems in d-degenerate graphs. *Discrete Applied Mathematics*, 160(15):2131–2141, 2012.

[97] Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994.

[98] Peter Damaschke. Fixed-parameter enumerability of cluster editing and related problems. *Theory of Computing Systems*, 46(2):261–283, 2010.

[99] Anuj Dawar, Martin Grohe, and Stephan Kreutzer. Locally excluding a minor. In *LICS 2007*, pages 270–279. IEEE Computer Society, 2007.

[100] Anuj Dawar and Stephan Kreutzer. Domination problems in nowhere-dense classes. In *FSTTCS 2009*, volume 4 of *LIPIcs*, pages 157–168. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009.

[101] Holger Dell and Dániel Marx. Kernelization of packing problems. In *SODA 2012*, pages 68–81. SIAM, 2012.

[102] Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *STOC 2010*, pages 251–260, 2010.

[103] Erik D. Demaine, Fedor V. Fomin, MohammadTaghi Hajiaghayi, and Dimitrios M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Transactions on Algorithms*, 1(1):33–47, 2005.

[104] Erik D. Demaine, Fedor V. Fomin, MohammadTaghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

[105] Erik D. Demaine and MohammadTaghi Hajiaghayi. Bidimensionality: New connections between FPT algorithms and PTASs. In *SODA 2005*, pages 590–601. SIAM, 2005.

[106] Erik D. Demaine and MohammadTaghi Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

[107] Erik D. Demaine and MohammadTaghi Hajiaghayi. Linearity of grid minors in treewidth with applications through bidimensionality. *Combinatorica*, 28(1):19–36, 2008.

[108] Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Algorithmic Graph Minor Theory: Decomposition, Approximation, and Coloring. In *FOCS 2005*, pages 637–646, 2005.

[109] Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Contraction decomposition in H-minor-free graphs and algorithmic applications. In *STOC 2011*, pages 441–450. ACM, 2011.

[110] Matt DeVos, Guoli Ding, Bogdan Oporowski, Daniel P. Sanders, Bruce A. Reed, Paul D. Seymour, and Dirk Vertigan. Excluding any graph as a minor allows a low tree-width 2-coloring. *J. Comb. Theory, Ser. B*, 91(1):25–41, 2004.

[111] Josep Díaz, Jordi Petit, and Maria J. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002.

[112] Reinhard Diestel. *Graph Theory*. Springer, 2005.

[113] Reinhard Diestel, Tommy R. Jensen, Konstantin Yu. Gorbunov, and Carsten Thomassen. Highly connected sets and the excluded grid theorem. *J. Comb. Theory, Ser. B*, 75(1):61–73, 1999.

[114] Irit Dinur and Shmuel Safra. The importance of being biased. In *STOC 2002*, pages 33–42. ACM, 2002.

[115] Michael Dom, Jiong Guo, Falk Hüffner, Rolf Niedermeier, and Anke Truß. Fixed-parameter tractability results for feedback set problems in tournaments. *J. Discrete Algorithms*, 8(1):76–86, 2010.

[116] Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Incompressibility through colors and IDs. In *ICALP 2009 (1)*, pages 378–389, 2009.

[117] Frederic Dorn. Dynamic programming and fast matrix multiplication. In *ESA 2006*, volume 4168 of *Lecture Notes in Computer Science*, pages 280–291. Springer, 2006.

[118] Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. Subexponential parameterized algorithms. *Computer Science Review*, 2(1):29–39, 2008.

[119] Rodney G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer-Verlag, New York, 1999.

[120] Andrew Drucker. New limits to classical and quantum instance compression. In *FOCS 2012*, pages 609–618. IEEE Computer Society, 2012.

[121] Zdenek Dvořák, Daniel Král, and Robin Thomas. Deciding first-order properties for sparse graphs. In *FOCS 2010*, pages 133–142. IEEE Computer Society, 2010.

[122] Pál Erdős. On an elementary proof of some asymptotic formulas in the theory of partitions. *Annals of Mathematics (2)*, 43:437–450, 1942.

[123] Pál Erdős, Adolph W. Goodman, and Lajos Pósa. The representation of a graph by set intersections. *Canadian Journal of Mathematics*, 18:106–112, 1966.

[124] Pál Erdős and George Szekeres. A combinatorial problem in geometry. *Compositio Math.*, 2:463–470, 1935.

[125] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976.

[126] Uriel Feige. Faster FAST (Feedback Arc Set in Tournaments). *CoRR*, abs/0911.5094, 2009.

[127] Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, 38(2):629–657, 2008.

[128] Michael R. Fellows, Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. Graph-based data clustering with overlaps. *Discrete Optimization*, 8(1):2–17, 2011.

[129] Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is NP-complete. *SIAM J. Discrete Math.*, 23(2):909–939, 2009.

[130] Henning Fernau, Fedor V. Fomin, Daniel Lokshtanov, Matthias Mnich, Geevarghese Philip, and Saket Saurabh. Ranking and drawing in subexponential time. In *IWOCA 2010*, volume 6460 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2010.

[131] Jörg Flum and Martin Grohe. Fixed-parameter tractability, definability, and model-checking. *SIAM J. Comput.*, 31(1):113–145, 2001.

[132] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

[133] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Algorithmic lower bounds for problems parameterized with clique-width. In *SODA 2010*, pages 493–502. SIAM, 2010.

[134] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010.

[135] Fedor V. Fomin and Kjartan Høie. Pathwidth of cubic graphs and exact algorithms. *Inf. Process. Lett.*, 97(5):191–196, 2006.

[136] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. An EATCS Series: Texts in Theoretical Computer Science. Springer, 2010.

[137] Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of Cluster Editing. In *STACS 2013*, volume 20 of *LIPIcs*, pages 32–43. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[138] Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar $\mathcal{F}$-Deletion: Approximation, kernelization and optimal FPT algorithms. In *FOCS 2012*, pages 470–479. IEEE Computer Society, 2012.

[139] Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Fast local search algorithm for weighted Feedback Arc Set in Tournaments. In *AAAI 2010*. AAAI Press, 2010.

[140] Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Bidimensionality and EPTAS. In *SODA 2011*, pages 748–759. SIAM, 2011.

[141] Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Subexponential algorithms for partial cover problems. *Inf. Process. Lett.*, 111(16):814–818, 2011.

[142] Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Bidimensionality and geometric graphs. In *SODA 2012*, pages 1563–1575. SIAM, 2012.

[143] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. In *SODA 2010*, pages 503–510. SIAM, 2010.

[144] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Linear kernels for (Connected) Dominating Set on H-minor-free graphs. In *SODA 2012*, pages 82–93. SIAM, 2012.

[145] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Linear kernels for (Connected) Dominating Set on graphs with excluded topological subgraphs. In *STACS 2013*, volume 20 of *LIPIcs*, pages 92–103. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[146] Fedor V. Fomin and Michał Pilipczuk. Jungles, bundles, and fixed-parameter tractability. In *SODA 2013*, pages 396–413. SIAM, 2013.

[147] Fedor V. Fomin and Michał Pilipczuk. Subexponential parameterized algorithm for computing the cutwidth of a semi-complete digraph. *CoRR*, abs/1301.7314, 2013. To appear in the proceedings of ESA 2013.

[148] Fedor V. Fomin and Dimitrios M. Thilikos. Dominating sets in planar graphs: Branch-width and exponential speed-up. *SIAM J. Comput.*, 36(2):281–309, 2006.

[149] Fedor V. Fomin and Yngve Villanger. Subexponential parameterized algorithm for minimum fill-in. In *SODA 2012*, pages 1737–1746. SIAM, 2012.

[150] Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.

[151] Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980.

[152] Alexandra Fradkin and Paul D. Seymour. Edge-disjoint paths in digraphs with bounded independence number, 2010. Manuscript.

[153] Alexandra Ovetsky Fradkin and Paul D. Seymour. Tournament pathwidth and topological containment. *J. Comb. Theory, Ser. B*, 103(3):374–384, 2013.

[154] Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001.

[155] Markus Frick and Martin Grohe. The complexity of First-Order and Monadic Second-Order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004.

[156] Haim Gaifman. On local and non-local properties. In *Logic Colloquium '81*, pages 105–135. North-Holland, 1982.

[157] Robert Ganian and Petr Hliněný. On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. *Discrete Applied Mathematics*, 158(7):851–867, 2010.

[158] Robert Ganian, Petr Hliněný, Joachim Kneis, Alexander Langer, Jan Obdržálek, and Peter Rossmanith. On digraph width measures in parameterized algorithmics. In *IWPEC 2009*, volume 5917 of *Lecture Notes in Computer Science*, pages 185–197. Springer, 2009.

[159] Robert Ganian, Petr Hliněný, Joachim Kneis, Daniel Meister, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Are there any good digraph width measures? In *IPEC 2010*, volume 6478 of *Lecture Notes in Computer Science*, pages 135–146. Springer, 2010.

[160] Robert Ganian, Petr Hliněný, and Jan Obdržálek. Clique-width: When hard does not mean impossible. In *STACS 2011*, volume 9 of *LIPIcs*, pages 404–415. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[161] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman, 1979.

[162] Michael R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[163] Michael R. Garey, David S. Johnson, and Robert E. Tarjan. The planar hamiltonian circuit problem is NP-complete. *SIAM J. Comput.*, 5(4):704–714, 1976.

[164] Jim Geelen and Sang-il Oum. Circle graph obstructions under pivoting. *Journal of Graph Theory*, 61(1):1–11, 2009.

[165] Esha Ghosh, Sudeshna Kolay, Mrinal Kumar, Pranabendu Misra, Fahad Panolan, Ashutosh Rai, and M. S. Ramanujan. Faster parameterized algorithms for deletion to split graphs. In *SWAT 2012*, volume 7357 of *Lecture Notes in Computer Science*, pages 107–118. Springer, 2012.

[166] Archontia C. Giannopoulou, Iosif Salem, and Dimitris Zoros. Effective computation of immersion obstructions for unions of graph classes. In *SWAT 2012*, volume 7357 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2012.

[167] Ioannis Giotis and Venkatesan Guruswami. Correlation clustering with a fixed number of clusters. In *SODA 2006*, pages 1167–1176. ACM Press, 2006.

[168] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, 2005.

[169] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Data reduction and exact algorithms for Clique Cover. *ACM Journal of Experimental Algorithmics*, 13, 2008.

[170] Jens Gramm, Jiong Guo, Falk Hüffner, Rolf Niedermeier, Hans-Peter Piepho, and Ramona Schmid. Algorithms for compact letter displays: Comparison and evaluation. *Computational Statistics & Data Analysis*, 52(2):725–736, 2007.

[171] Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for Closest String and related problems. *Algorithmica*, 37(1):25–42, 2003.

[172] David A. Gregory and Norman J. Pullman. On a clique covering problem of Orlin. *Discrete Mathematics*, 41(1):97–99, 1982.

[173] Martin Grohe. Computing crossing numbers in quadratic time. *J. Comput. Syst. Sci.*, 68(2):285–302, 2004.

[174] Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *STOC 2011*, pages 479–488. ACM, 2011.

[175] Martin Grohe, Ken-ichi Kawarabayashi, and Bruce A. Reed. A simple algorithm for the graph minor decomposition - logic meets structural graph theory. In *SODA 2013*, pages 414–431. SIAM, 2013.

[176] Martin Grohe and Stephan Kreutzer. Methods for algorithmic meta-theorems. *Contemporary Mathematics*, 588, 2011.

[177] Martin Grohe and Dániel Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. In *STOC 2012*, pages 173–192. ACM, 2012.

[178] Jonathan L. Gross and Jay Yellen. *Graph Theory and its Applications*. CRC Press, 2006.

[179] Qian-Ping Gu and Hisao Tamaki. Improved bounds on the planar branchwidth with respect to the largest grid minor size. *Algorithmica*, 64(3):416–453, 2012.

[180] Jean-Loup Guillaume and Matthieu Latapy. Bipartite structure of all complex networks. *Inf. Process. Lett.*, 90(5):215–221, 2004.

[181] Sylvain Guillemot and Matthias Mnich. Kernel and fast algorithm for dense triplet inconsistency. In *TAMC 2010*, volume 6108 of *Lecture Notes in Computer Science*, pages 247–257. Springer, 2010.

[182] Jiong Guo. A more effective linear kernelization for Cluster Editing. *Theoretical Computer Science*, 410(8-10):718–726, 2009.

[183] Jiong Guo, Iyad A. Kanj, Christian Komusiewicz, and Johannes Uhlmann. Editing graphs into disjoint unions of dense clusters. *Algorithmica*, 61(4):949–970, 2011.

[184] Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. A more relaxed model for graph-based data clustering: s-plex Cluster Editing. *SIAM J. Discrete Math.*, 24(4):1662–1683, 2010.

[185] Frank Gurski. A comparison of two approaches for polynomial time algorithms computing basic graph parameters. *CoRR*, abs/0806.4073, 2008.

[186] Godfrey H. Hardy and Srinivasa Ramanujan. Asymptotic formulae in combinatory analysis. *Proceedings of the London Mathematical Society*, s2-17(1):75–115, 1918.

[187] David Hartvigsen. The planar Multiterminal Cut problem. *Discrete Applied Mathematics*, 85(3):203–222, 1998.

[188] Danny Hermelin and Xi Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *SODA 2012*, pages 104–113. SIAM, 2012.

[189] Petr Hliněný and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008.

[190] Petr Hliněný, Sang-il Oum, Detlef Seese, and Georg Gottlob. Width parameters beyond tree-width and their applications. *Comput. J.*, 51(3):326–362, 2008.

[191] D. N. Hoover. Complexity of graph covering problems for graphs of low degree. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 11:187–208, 1992.

[192] Wen-Lian Hsu and Kuo-Hui Tsai. Linear time algorithms on circular-arc graphs. *Inf. Process. Lett.*, 40(3):123–129, 1991.

[193] Falk Hüffner. Algorithm engineering for optimal graph bipartization. *J. Graph Algorithms Appl.*, 13(2):77–98, 2009.

[194] Paul Hunter and Stephan Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theoretical Computer Science*, 399(3):206–219, 2008.

[195] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[196] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[197] Jisu Jeong, O-joung Kwon, and Sang-il Oum. Excluded vertex-minors for graphs of linear rank-width at most k. In *STACS 2013*, volume 20 of *LIPIcs*, pages 221–232. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[198] Thor Johnson, Neil Robertson, Paul D. Seymour, and Robin Thomas. Directed tree-width. *J. Combin. Theory Ser. B*, 82(1):138–154, 2001.

[199] Gwenaël Joret and David R. Wood. Complete graph minors and the graph minor structure theorem. *J. Comb. Theory, Ser. B*, 103(1):61–74, 2013.

[200] Marcin Kamiński and Naomi Nishimura. Finding an induced path of given parity in planar graphs in polynomial time. In *SODA 2012*, pages 656–670. SIAM, 2012.

[201] Marcin Kamiński and Dimitrios M. Thilikos. Contraction checking in graphs on surfaces. In *STACS 2012*, volume 14 of *LIPIcs*, pages 182–193. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[202] Mamadou Moustapha Kanté and Michael Rao. The rank-width of edge-colored graphs. *CoRR*, abs/0709.1433, 2007.

[203] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[204] Richard M. Karp. On the complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

[205] Marek Karpinski and Warren Schudy. Faster algorithms for Feedback Arc Set Tournament, Kemeny Rank Aggregation and Betweenness Tournament. In *ISAAC 2010*, volume 6506 of *Lecture Notes in Computer Science*, pages 3–14. Springer, 2010.

[206] Ken-ichi Kawarabayashi. Planarity allowing few error vertices in linear time. In *FOCS 2009*, pages 639–648. IEEE Computer Society, 2009.

[207] Ken-ichi Kawarabayashi and Yusuke Kobayashi. Linear min-max relation between the treewidth of H-minor-free graphs and its largest grid. In *STACS 2012*, volume 14 of *LIPIcs*, pages 278–289. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[208] Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce A. Reed. The disjoint paths problem in quadratic time. *J. Comb. Theory, Ser. B*, 102(2):424–435, 2012.

[209] Ken-ichi Kawarabayashi and Bruce A. Reed. Computing crossing number in linear time. In *STOC 2007*, pages 382–390. ACM, 2007.

[210] Ken-ichi Kawarabayashi and Bruce A. Reed. Hadwiger's conjecture is decidable. In *STOC 2009*, pages 445–454. ACM, 2009.

[211] Ken-ichi Kawarabayashi and Bruce A. Reed. An (almost) linear time algorithm for odd cyles transversal. In *SODA 2010*, pages 365–378. SIAM, 2010.

[212] Ken-ichi Kawarabayashi and Paul Wollan. A simpler algorithm and shorter proof for the graph minor decomposition. In *STOC 2011*, pages 451–458. ACM, 2011.

[213] Eduardo Kellerman. Determination of keyword conflict. *IBM Technical Disclosure Bulletin*, 16(2):544–546, 1973.

[214] Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *STOC 2007*, pages 95–103. ACM, 2007.

[215] Denés Kőnig. Gráfok és mátrixok. *Matematikai és Fizikai Lapok*, 38:116–119, 1931. In Hungarian.

[216] Ilhee Kim and Paul D. Seymour. Tournament minors. *CoRR*, abs/1206.3135, 2012.

[217] Philip N. Klein and Dániel Marx. Solving Planar k-Terminal Cut in $O(n^{c\sqrt{k}})$ time. In *ICALP 2012 (1)*, volume 7391 of *Lecture Notes in Computer Science*, pages 569–580. Springer, 2012.

[218] Jon M. Kleinberg and Éva Tardos. *Algorithm design*. Addison-Wesley, 2006.

[219] Joachim Kneis and Alexander Langer. A practical approach to Courcelle's theorem. *Electr. Notes Theor. Comput. Sci.*, 251:65–81, 2009.

[220] Joachim Kneis, Alexander Langer, and Peter Rossmanith. Courcelle's theorem - a game-theoretic approach. *Discrete Optimization*, 8(4):568–594, 2011.

[221] Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic Feedback Vertex Set. *CoRR*, abs/1306.3566, 2013.

[222] Christian Komusiewicz and Johannes Uhlmann. Cluster Editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012.

[223] Lawrence T. Kou, Larry J. Stockmeyer, and C. K. Wong. Covering edges by cliques with regard to keyword conflicts and intersection graphs. *Communications of the ACM*, 21(2):135–139, 1978.

[224] Łukasz Kowalik and Marcin Mucha. A $9k$ kernel for Nonseparating Independent Set in planar graphs. In *WG 2012*, volume 7551 of *Lecture Notes in Computer Science*, pages 160–171. Springer, 2012.

[225] Stefan Kratsch and Magnus Wahlström. Compression via matroids: a randomized polynomial kernel for odd cycle transversal. In *SODA 2012*, pages 94–103. SIAM, 2012.

[226] Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *FOCS 2012*, pages 450–459. IEEE Computer Society, 2012.

[227] Stephan Kreutzer. Presentation during Dagstuhl Seminar *Bidimensional Structures: Algorithms, Combinatorics and Logic* (13121).

[228] Stephan Kreutzer. On the parameterized intractability of Monadic Second-Order logic. *Logical Methods in Computer Science*, 8(1), 2012.

[229] Stephan Kreutzer and Anuj Dawar. Parameterized complexity of First-Order logic. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:131, 2009.

[230] Stephan Kreutzer and Sebastian Ordyniak. Digraph decompositions and monotonicity in digraph searching. *Theoretical Computer Science*, 412(35):4688–4703, 2011.

[231] Stephan Kreutzer and Siamak Tazari. Lower bounds for the complexity of Monadic Second-Order logic. In *LICS 2010*, pages 189–198. IEEE Computer Society, 2010.

[232] Stephan Kreutzer and Siamak Tazari. On brambles, grid-like minors, and parameterized intractability of Monadic Second-Order logic. In *SODA 2010*, pages 354–364. SIAM, 2010.

[233] Joseph B. Kruskal. Well-quasi-ordering, the tree theorem and Vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95:210–225, 1960.

[234] Kazimierz Kuratowski. Sur le problème des courbes gauches en Topologie. *Fundamenta Mathematicae*, 15(1):271–283, 1930. In French.

[235] Jens Lagergren. Efficient parallel algorithms for tree-decomposition and related problems. In *FOCS 1990*, pages 173–182. IEEE Computer Society, 1990.

[236] Michael Lampis, Georgia Kaouri, and Valia Mitsou. On the algorithmic effectiveness of digraph decompositions and complexity measures. *Discrete Optimization*, 8(1):129–138, 2011.

[237] Alexander Leaf and Paul D. Seymour. Treewidth and planar minors, 2012. Manuscript.

[238] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1979.

[239] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In *SODA 2011*, pages 777–789, 2011.

[240] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.

[241] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. In *SODA 2011*, pages 760–776, 2011.

[242] Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization - preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond*, volume 7370 of *Lecture Notes in Computer Science*, pages 129–161. Springer, 2012.

[243] Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *CoRR*, abs/1203.0833, 2012.

[244] Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In *STOC 2010*, pages 321–330. ACM, 2010.

[245] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.

[246] Bin Ma and Xiaoming Sun. More efficient algorithms for Closest String and substring problems. *SIAM J. Comput.*, 39(4):1432–1443, 2009.

[247] Shaohan Ma, Walter D. Wallis, and Julin Wu. Clique covering of chordal graphs. *Utilitas Mathematica*, 36:151–152, 1989.

[248] Dániel Marx. On the optimality of planar and geometric approximation schemes. In *FOCS 2007*, pages 338–348. IEEE Computer Society, 2007.

[249] Dániel Marx. Closest substring problems with small distances. *SIAM J. Comput.*, 38(4):1382–1410, 2008.

[250] Dániel Marx. Can you beat treewidth? *Theory of Computing*, 6(1):85–112, 2010.

[251] Dániel Marx. A tight lower bound for Planar Multiway Cut with fixed number of terminals. In *ICALP 2012 (1)*, volume 7391 of *Lecture Notes in Computer Science*, pages 677–688. Springer, 2012.

[252] Dániel Marx. What's next? Future directions in parameterized complexity. In *The Multivariate Algorithmic Revolution and Beyond*, volume 7370 of *Lecture Notes in Computer Science*, pages 469–496. Springer, 2012.

[253] Dániel Marx and Ildikó Schlotter. Obtaining a planar graph by vertex deletion. *Algorithmica*, 62(3-4):807–822, 2012.

[254] Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*. Johns Hopkins series in the mathematical sciences. Johns Hopkins University Press, 2001.

[255] N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. LP can be a cure for parameterized problems. In *STACS 2012*, volume 14 of *LIPIcs*, pages 338–349. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[256] Crispin Nash-Williams. On well-quasi-ordering finite trees. *Proc. Cambridge Philos. Soc.*, 59:833–835, 1963.

[257] Jesper Nederlof. Fast polynomial-space algorithms using inclusion-exclusion. *Algorithmica*, 65(4):868–884, 2013.

[258] Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

[259] Jan Obdržálek. DAG-width: connectivity measure for directed graphs. In *SODA 2006*, pages 814–821. ACM Press, 2006.

[260] James B. Orlin. Contentment in graph theory: Covering graphs with cliques. *Indagationes Mathematicae (Proceedings)*, 80(5):406–424, 1977.

[261] Sang-il Oum. Rank-width and vertex-minors. *J. Comb. Theory, Ser. B*, 95(1):79–100, 2005.

[262] Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Transactions on Algorithms*, 5(1), 2008.

[263] Sang-il Oum. Rank-width and well-quasi-ordering. *SIAM J. Discrete Math.*, 22(2):666–682, 2008.

[264] Sang-il Oum. Rank-width and well-quasi-ordering of skew-symmetric or symmetric matrices (extended abstract). *Electronic Notes in Discrete Mathematics*, 38:693–698, 2011.

[265] Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

[266] Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364, 2005.

[267] Hans-Peter Piepho. An algorithm for a letter-based representation of all-pairwise comparisons. *Journal of Computational and Graphical Statistics*, 13(2):456–466, 2004.

[268] Michał Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In *MFCS 2011*, volume 6907 of *Lecture Notes in Computer Science*, pages 520–531. Springer, 2011.

[269] Michał Pilipczuk. Computing cutwidth and pathwidth of semi-complete digraphs via degree orderings. In *STACS 2013*, volume 20 of *LIPIcs*, pages 197–208. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[270] Michael D. Plummer and László Lovász. *Matching Theory*. ACM Chelsea Publishing, Providence, Rhode Island, 2009.

[271] Fábio Protti, Maise Dantas da Silva, and Jayme Luiz Szwarcfiter. Applying modular decomposition to parameterized cluster editing problems. *Theory of Computing Systems*, 44(1):91–104, 2009.

[272] Mihai Pătraşcu and Ryan Williams. On the possibility of faster SAT algorithms. In *SODA 2010*, pages 1065–1075. SIAM, 2010.

[273] Subramanian Rajagopalan, Manish Vachharajani, and Sharad Malik. Handling irregular ILP within conventional VLIW schedulers using artificial resource constraints. In *CASES 2000*, pages 157–164, 2000.

[274] Venkatesh Raman and Saket Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theoretical Computer Science*, 351(3):446–458, 2006.

[275] Bruce A. Reed. Finding approximate separators and computing tree width quickly. In *STOC 1992*, pages 221–228. ACM, 1992.

[276] Bruce A. Reed. Introducing directed tree width. *Electronic Notes in Discrete Mathematics*, 3:222–229, 1999.

[277] Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004.

[278] Fred S. Roberts. Applications of edge coverings by cliques. *Discrete Applied Mathematics*, 10(1):93 – 109, 1985.

[279] Neil Robertson and Paul D. Seymour. Graph Minors. I. Excluding a forest. *J. Comb. Theory, Ser. B*, 35(1):39–61, 1983.

[280] Neil Robertson and Paul D. Seymour. Graph Minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

[281] Neil Robertson and Paul D. Seymour. Graph Minors. V. Excluding a planar graph. *J. Comb. Theory, Ser. B*, 41(1):92–114, 1986.

[282] Neil Robertson and Paul D. Seymour. Graph Minors. IV. Tree-width and well-quasi-ordering. *J. Comb. Theory, Ser. B*, 48(2):227–254, 1990.

[283] Neil Robertson and Paul D. Seymour. Graph Minors. VIII. A Kuratowski theorem for general surfaces. *J. Comb. Theory, Ser. B*, 48(2):255–288, 1990.

[284] Neil Robertson and Paul D. Seymour. Graph Minors. X. Obstructions to tree-decomposition. *J. Comb. Theory, Ser. B*, 52(2):153–190, 1991.

[285] Neil Robertson and Paul D. Seymour. Graph Minors. XIII. The Disjoint Paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

[286] Neil Robertson and Paul D. Seymour. Graph Minors. XVI. Excluding a non-planar graph. *J. Comb. Theory, Ser. B*, 89(1):43–76, 2003.

[287] Neil Robertson and Paul D. Seymour. Graph Minors. XX. Wagner's conjecture. *J. Comb. Theory, Ser. B*, 92(2):325–357, 2004.

[288] Neil Robertson and Paul D. Seymour. Graph Minors XXIII. Nash-Williams' immersion conjecture. *J. Comb. Theory, Ser. B*, 100(2):181–205, 2010.

[289] Neil Robertson and Paul D. Seymour. Graph Minors. XXII. Irrelevant vertices in linkage problems. *J. Comb. Theory, Ser. B*, 102(2):530–563, 2012.

[290] Neil Robertson, Paul D. Seymour, and Robin Thomas. Quickly excluding a planar graph. *J. Comb. Theory, Ser. B*, 62(2):323–348, 1994.

[291] John M. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7(3):425–440, 1986.

[292] Mohammad Ali Safari. D-width: A more natural measure for directed tree width. In *MFCS 2005*, volume 3618 of *Lecture Notes in Computer Science*, pages 745–756. Springer, 2005.

[293] Alexander Schrijver. Finding k disjoint paths in a directed planar graph. *SIAM J. Comput.*, 23(4):780–788, 1994.

[294] Detlef Seese. The structure of models of decidable monadic theories of graphs. *Ann. Pure Appl. Logic*, 53(2):169–195, 1991.

[295] Detlef Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996.

[296] Maria J. Serna and Dimitrios M. Thilikos. Parameterized complexity for graph layout problems. *Bulletin of the EATCS*, 86:41–65, 2005.

[297] Paul D. Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *J. Comb. Theory, Ser. B*, 58(1):22–33, 1993.

[298] Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[299] Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.

[300] Aleksandrs Slivkins. Parameterized tractability of edge-disjoint paths on directed acyclic graphs. *SIAM J. Discrete Math.*, 24(1):146–157, 2010.

[301] Hisao Tamaki. A polynomial time algorithm for bounded directed pathwidth. In *WG 2011*, volume 6986 of *Lecture Notes in Computer Science*, pages 331–342. Springer, 2011.

[302] Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Cutwidth I: A linear time fixed parameter algorithm. *J. Algorithms*, 56(1):1–24, 2005.

[303] Stéphan Thomassé. A $4k^2$ kernel for Feedback Vertex Set. *ACM Transactions on Algorithms*, 6(2), 2010.

[304] Patrick Traxler. The time complexity of constraint satisfaction. In *IWPEC 2008*, volume 5018 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 2008.

[305] Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009.

[306] Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001.

[307] Klaus Wagner. Über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114(1):570–590, 1937. In German.

[308] The parameterized complexity community wiki. http://fpt.wikidot.com/.

[309] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[310] David Zuckerman. Linear degree extractors and the inapproximability of Max Clique and Chromatic Number. In *STOC 2006*, pages 681–690. ACM, 2006.

Acknowledgements

First of all, I would like to thank my supervisor Fedor Vladimirovich Fomin. I am grateful not only for the excellent scientific guidance, in particular for suggesting the topic of topological problems in tournaments which turned out to be extremely fruitful, but also for all our friendly discussions about computer science, academic community, life in Norway and life in general. It is rare to have a supervisor on whose door you can knock any time of the day, and who will be always happy to talk about your next project, every time calling it 'cool'. Thanks a lot, Fedor!

Many thanks go also to my co-supervisor, Pinar Heggernes, whose help and advice was priceless in many different situations. What I particularly appreciate, is that this help and advice always come with the never-vanishing smile and sympathy.

Second, I would like to thank in advance my committee members for all the hard work put into reading and understanding the presented mathematics. I hope that the volume of this thesis scared you only a little when you have first received it.

Next, I would like to thank all of my co-authors: Ivan Bliznets, Hans L. Bodlaender, Rajesh Chitnis, Marek Cygan, Pål Grønås Drange, Markus Sortland Dregi, Fedor V. Fomin, Archontia C. Giannopoulou, Petr A. Golovach, MohammadTaghi Hajiaghayi, Pinar Heggernes, Pim van 't Hof, Bart M. P. Jansen, Stefan Kratsch, Erik Jan van Leeuwen, Daniel Lokshtanov, Fredrik Manne, Dániel Marx, Jesper Nederlof, Daniël Paulusma, Geevarghese Philip, Marcin Pilipczuk, Johan M. M. van Rooij, Piotr Sankowski, Saket Saurabh, Ildikó Schlotter, Riste Škrekovski, Yngve Villanger, Magnus Wahlström, and Jakub Onufry Wojtaszczyk. Working with all of you was a real pleasure, and the fact that I could contribute to all the great ideas that came out of our collaboration fills me with pride. Special thanks are due to Marcin and Marek, with whom I have not only shared many projects, but on whom I could always count when a critical advice on anything was needed.

Many warm thanks go also to the Algorithms Group in Bergen. The amazing family atmosphere on the 3rd floor creates a unique research environment that is at the same time relaxing and extremely stimulating. It may sound a little bit strange, but coming to work has been really a great pleasure for me during these two years.

Next, I would like to express my thanks to all the friends that I have met in Norway. Your presence filled my last two years here with life, and I have shared with you many unforgettable moments. Thanks to you, I will always feel in Bergen like at home.

Of course, warm thanks go also to all the friends back in Poland. It is still amazing for me that every time when I come back to Warsaw for some period, I can spend a couple of hours at the faculty just talking to all the friends as if it had been just a few days of my absence. It makes me really happy that when eventually returning to Warsaw, I will actually have a lot to return to.

Special thanks are due to my parents, for all the love, support, and advice with which you helped me during all these years. This thesis is in many different aspects also a result of your hard work and efforts; I hope you will be proud of it.

Last but not least, I would like to thank the European Research Council for supporting my research via ERC Grant "Rigorous Theory of Preprocessing", no. 267959. I acknowledge also the financial support of Meltzerfondet.

Michał Pilipczuk,
August 2013.