Algorithmic Graph Minor Theory: Approximation, Parameterized Complexity, and Practical Aspects

DISSERTATION

zur Erlangung des akademischen Grades
DOCTOR RERUM NATURALIUM
im Fach Informatik

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

von
Dipl.-Math. Siamak Tazari
geboren am 1.9.1982 in Teheran, Iran

Präsident der Humboldt-Universität zu Berlin:
Prof. Dr. Dr. h.c. Christoph Markschies

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät II:
Prof. Dr. Peter Frensch

Gutachter:
1. Prof. Dr. Martin Grohe
2. Prof. Dr. Matthias Müller-Hannemann
3. Prof. Philip Klein, Ph.D.

eingereicht am: 26.5.2010
Tag der Verteidigung: 29.7.2010

Zusammenfassung

In der theoretischen Informatik, unterscheiden wir zwischen effizient lösbaren und rechnerisch schweren Problemen. Aber auch schwere Probleme müssen gelöst werden und es gibt verschiedene Methoden, um mit ihnen umzugehen. Eine dieser Methoden besteht darin, die Eingabe auf bestimmte Instanzklassen zu beschränken, die algorithmisch besser handhabbar sind. Bei graphentheoretischen Problemen, entspricht dies dem Ansatz, spezielle Graphenklassen zu betrachten, wie zum Beispiel planare Graphen oder Graphen, die auf einer gegebenen festen Fläche einbettbar sind. Eine weitere Verallgemeinerung solcher Graphen führt zu Klassen von Graphen, die unter Minorenbildung abgeschlossen sind. Minorenabgeschlossene Klassen von Graphen haben in den letzten Jahrzehnten an enormer Bedeutung gewonnen, nicht zuletzt wegen der Graphenminorentheorie von Robertson und Seymour. In dieser Dissertation betrachten wir verschiedene Algorithmen und Komplexitätstheoretische Resultate, die für minorenabgeschlossene Klassen gelten oder eng mit der Minorentheorie verwandt sind. Die Arbeit besteht aus drei Teilen.

Im ersten Teil, betrachten wir das Steinerbaumproblem in eingebetteten Graphen. Dies ist eines der bedeutendsten Probleme in der Informatik, sowohl in der Theorie als auch in der Praxis, und diente stets als eines der Standardprobleme, auf dem neue algorithmische Ideen entwickelt und untersucht worden sind. Vor kurzem wurde ein polynomielles Approximationsschema (PTAS) für dieses Problem in planaren Graphen von Borradaile, Klein und Mathieu entwickelt (2007,2009). Wir (i) zeigen, wie dieser PTAS von planaren Graphen auf Graphen mit beschränktem Genus erweitert werden kann; (ii) bieten eine Implementierung und Engineering des Algorithmus an und zeigen, dass es sogar auf großen Instanzen erstaunlich gut funktioniert; und (iii) zeigen, wie dieser Algorithmus angewendet werden kann, um einen PTAS für das geometrische Steinerbaumproblem mit Hindernissen in der Ebene zu erhalten.

Im zweiten Teil der Arbeit, betrachten wir Algorithmen auf generellen echten minorenabgeschlossenen Klassen von Graphen. Zum einen, entwickeln wir einen kürzeste Wege Algorithmus mit linearer Laufzeit auf diesen Klassen, in dem wir einen entsprechenden Algorithmus für planare Graphen von Henzinger, Klein, Rao und Subramanian (1994,1997) erweitern; daraus entwickeln wir ebenfalls eine 2-Approximation für das Steinerbaumproblemen in Linearzeit auf diesen Klassen. Zum anderen, zeigen wir, wie man eine große Anzahl von Approximationsschemata und parametrischen Algorithmen auf solchen Klassen wesentlich beschleunigen kann.

Im letzten Teil, betrachten wir die Komplexität des Model-Checking Problems der monadischen Logik zweiter Stufe (MSO) mit Quantifizierung über Kanten: gegeben eine feste MSO-Formel und ein Graph, möchten wir überprüfen, ob die Formel in dem Graphen gilt. Ein sehr bekannter Satz von Courcelle (1990) besagt, dass dieses Problem auf Graphenklassen mit beschränkter Baumweite effizient lösbar ist. Wir präsentieren erste untere Schranken für dieses Problem, in dem Sinne, dass wir zeigen, dass es Klassen von Graphen gibt, die unbeschränkte, aber sehr kleine, Baumweite haben und in denen das Model-Checking Problem von MSO nicht effizient lösbar ist (unter angemessenen Voraussetzungen aus der Komplexitätstheorie). Um dies zu erreichen, entwickeln wir mehrere Algorithmen, um einige maßgebliche Strukturen aus der Graphenstrukturen- und minorentheorie effizient zu berechnen. Unter anderem zeigen wir, wie Brambles, Gitter-ähnliche Minoren und bestimmte Baumgeordnete Gewebe in generellen Graphen mit ausreichend hoher Baumweite effizient konstruiert werden können.

Abstract

Theoretical computer science provides us with tools and techniques to classify problems as efficiently solvable or computationally hard. But even hard problems have to be solved and there exist various ways to attack them. One way is to restrict the input to certain classes that are computationally more feasible. In the case of graph-theoretic problems, this is akin to considering the problem on special graph classes. One of the most important such classes is the class of planar graphs or more generally, graphs that are embedded on a fixed surface; an even further generalization is to consider classes of graphs that are closed under taking minors. Such classes of graphs have been extensively studied in the past three decades, especially due to the deep graph minor theory of Robertson and Seymour. In this thesis, we consider various algorithms on or related to such graph classes and the theory of graph minors. The thesis consists of three parts:

In the first part, we consider the Steiner tree problem in embedded graphs. The Steiner tree problem is one of the most well-studied problems in computer science and a testbed for many new algorithmic ideas. Very recently, a polynomial-time approximation scheme (PTAS) for Steiner tree in planar graphs has been discovered by Borradaile, Klein, and Mathieu (2007,2009). We present an extension, engineering, and application of this algorithm: (i) we show how to generalize the technique of the PTAS from planar graphs to graphs of bounded genus and further subset connectivity problems; (ii) we implemented and engineered this algorithm from a practical point of view and show that it works surprisingly well, even on very large instances; and (iii) we apply this algorithm to obtain an efficient PTAS for the geometric Steiner tree problem among obstacles in the plane, which had been an open problem for a long time.

In the second part of the thesis, we consider further algorithms on minor-closed graph classes. We present a linear-time shortest-paths algorithm by generalizing the algorithm of Henzinger, Klein, Rao, and Subramanian (1994,1997) from planar graphs to all proper minor-closed graph classes; and we show how to obtain faster approximation schemes and parameterized algorithms for a large number of problems on these graph classes.

Finally, in the last part, we study the complexity of checking if a graph satisfies a formula of monadic second-order logic (MSO) with edge quantification when parameterized by the length of the formula. A famous theorem of Courcelle (1990) shows that this problem is efficiently solvable, that is, fixed-parameter tractable, on graphs of bounded treewidth. We provide first lower bounds on classes of colored graphs and subgraph-closed classes by showing that if the treewidth of such a class is not polylogarithmically bounded, the model-checking problem of MSO is not fixed-parameter tractable on the class (modulo some complexity assumptions). To achieve this, we develop several algorithmic tools using recent graph structure and graph minor theory, which are interesting in their own right. Most notably, we provide polynomial-time algorithms to construct brambles, grid-like minors, and tree-ordered webs in general graphs of sufficiently high treewidth.

Acknowledgement

First of all, I would like to thank my supervisors, Martin Grohe and Matthias Müller-Hannemann. In my first two years as a graduate student in Darmstadt, Matthias lead me gradually to the world of professional research in algorithmics; and when I came to Berlin afterwards, Martin guided me towards the great depths of matters in theoretical computer science. The immense amount of knowledge and know-how that I gained in these years and that finally led to the completion of this thesis is invaluable to me.

I would like to thank my further coauthors Glencora Borradaile, Erik Demaine, and Stephan Kreutzer for the exciting work we did together and for what I learned from them during these collaborations. Furthermore, each of these persons has had their unique influence on my vision of becoming a researcher and growing to be their colleague in the future; especially Stephan's role meant a lot to me in this respect.

I thank Philip Klein, whose research builds the foundation of a large part of this work, for serving on my committee. Furthermore, I thank Fedor Fomin and Dániel Marx for several helpful discussions; and also the numerous other exceptional researchers that I met, talked to, and learned from throughout these years.

I would like to thank the algorithmics group in TU-Darmstadt and the logic in computer science group in HU-Berlin for their continuous support, discussions, comments, and the time we spent together working and learning from each other. Particular thanks for proofreading parts of this thesis go to Paul Bonsma, Holger Dell, Kord Eickmeyer, Berit Grußien, André Hernich, Bastian Laubner, and Marc Thurley; also to Glencora Borradaile and Stephan Kreutzer; and to the anonymous reviewers of the parts of the thesis that have been also published otherwise.

I am grateful to the Deutsche Forschungsgemeinschaft (DFG), the Studienstiftung des Deutschen Volkes, the Berlin Mathematical School, and the graduate research training group "Methods for Discrete Structures" for both the financial and the other support they offered me during my time as a graduate student. The seminars, meetings, and guidance that I obtained through these institutions and the people involved in them contributed invaluably to my development as a researcher.

Last but not least, I would like to thank my family and my friends who kept my mind healthy and sane and gave me the energy and strength to keep on keeping on through all these years – in fact, through all my life.

Crimson flames tied through my ears
Rollin' high and mighty traps
Pounced with fire on flaming roads
Using ideas as my maps
"We'll meet on edges, soon," said I
Proud 'neath heated brow
Ah, but I was so much older then
I'm younger than that now

– Bob Dylan, 1964

To my parents, Mojgan and Saied,
for their unlimited love and support

Contents

Introduction

In theoretical computer science, we distinguish between problems that are *efficiently solvable* and *computationally hard* ones. The former are defined as the set of problems that admit a *polynomial-time* algorithm, usually referred to as P, and the latter is the class of NP-*hard* problems. It turns out that unless $\mathsf{P} = \mathsf{NP}$, an abundant number of important problems are computationally hard – but this does not obviate the need of solving them *somehow*; indeed, there exist various ways to attack such problems and one way is to *restrict the input* to certain classes that are computationally more feasible: oftentimes, a problem is hard in its full generality but turns out to be much better tractable on instances that are relevant in a certain context. In this thesis, we study the complexity of problems on restricted input classes. In some cases, we provide (more) efficient algorithms for the considered class, and in other cases, show the hardness of problems even on the constrained input.

One way to restrict the input is to consider a given problem together with a *parameter*. Sometimes a problem comes with a natural parameter – the simplest one being the size of the solution – such that if the parameter is small, the problem is tractable. This point of view leads to the theory of *parameterized complexity* [DF99, FG06] and we will often resort to this paradigm in this work.

Another common method to attack hard problems is the development of *approximation algorithms*: an algorithm that delivers a solution that is guaranteed to be at most a certain factor worse than the optimum. Ideally, we would like to have *arbitrarily close* approximations, that is, an algorithm that, for a given problem instance and $\varepsilon > 0$, finds a solution that is at most by a factor of $(1+\varepsilon)$ away from optimum. When dealing with a hard problem, the running time of such an algorithm must deteriorate with smaller ε and usually is exponential in ε^{-1}. We call such an algorithm a *polynomial-time approximation scheme* (PTAS) if its running time is polynomial for every fixed ε. Unfortunately, it turns out that lots of problems do not even admit a PTAS unless $\mathsf{P} = \mathsf{NP}$. Hence, we are often forced to combine these ideas: try to obtain an approximation scheme on restricted input. Indeed, a large portion of this thesis deals exactly with this matter.

A considerable number of computational problems are defined in or can be translated to the language of *graph theory*. In the case of graph theoretic-problems, restricting the input is akin to considering the problem on special graph classes. One of the most important of such classes is the class of *planar graphs*, graphs that can be drawn in the plane such that their edges do not cross. More generally, one can consider graphs that are *embedded on a fixed surface*, such as the torus or the Klein bottle. An even further generalization is to consider classes of graphs that are *closed under taking minors*, that is, closed under vertex and edge deletion and edge contraction. Such classes of graphs have been extensively studied in the past three decades, especially due to the deep graph

minor theory of Robertson and Seymour [RS04]. Besides proving some of the most seminal results in this area, Robertson and Seymour develop a vast toolbox of ideas and concepts to exploit the structure of graphs. In this work, we consider various algorithms on or related to such graph classes and the theory of graph minors. The thesis consists of the three parts outlined below.

Part I: Steiner Tree in Embedded Graphs: Extension, Engineering, and Application of a PTAS

The *Steiner tree* problem is one of the most fundamental problems in computer science and serves as a testbed for many new algorithmic ideas - both in theory and practice: given a graph and a subset of its vertices, called terminals, we wish to find the shortest tree in the graph that interconnects the terminals. The applications reach from all kinds of network design, perhaps most importantly VLSI design, to phylogenetic trees in bioinformatics [FG82, HRW92, KR95, CKM$^+$98, CD01]. The problem is well-known to be NP-hard [Kar72], even on planar graphs [GJ77], and does not even admit a PTAS in general graphs [BP89] unless P = NP. However, recently a PTAS for this problem on planar graphs has been discovered by Borradaile, Klein, and Mathieu [BKM09]. After a thorough introduction to the Steiner tree problem and its PTAS on planar graphs in Chapter 1, we have three chapters that deal with this PTAS, each from a different point of view. Chapter 2 is based on joint work with Glencora Borradaile and Erik Demaine [BDT09]; Chapters 3 and 4 are based on joint work with Matthias Müller-Hannemann [TM09a, MT07, MT10].

In Chapter 2, we present an *extension* of the work of [BKM09] by developing the first PTAS for Steiner tree in edge-weighted graphs of bounded genus. Our algorithm runs in time $\mathcal{O}(n \log n)$ for graphs embedded on both orientable and nonorientable surfaces with a constant that is singly exponential in the genus and the inverse of the desired accuracy. This work generalizes the PTAS frameworks of [BKM09, Kle06] from planar graphs to bounded-genus graphs, also in the sense that any future problems shown to admit the required structure theorem for planar graphs will similarly extend to bounded-genus graphs. In particular, this gives rise to PTASes in bounded-genus graphs for the subset traveling salesman problem [BDT09], $\{0, 1, 2\}$-edge-connected survivable network problem [BK08b], and also for the Steiner forest problem [BHM10].

In Chapter 3, we present the first attempt on *implementing and engineering* a highly theoretical polynomial-time approximation scheme with huge hidden constants, namely, the aforementioned PTAS for Steiner tree in planar graphs by Borradaile, Klein, and Mathieu. Whereas this result, and several other PTAS results of the recent years, are of high theoretical importance, no practical applications or even implementation attempts have been known to date due to the extremely large constants that are involved in them. We describe techniques on how to circumvent the challenges in implementing such a scheme. With today's limitations on processing power and space, we still have to sacrifice approximation guarantees for improved running times by choosing some parameters empirically. But our experiments show that with our choice of parameters,

we do get the desired approximation ratios, suggesting that a much tighter analysis might be possible.

Our computational experiments with benchmark instances from SteinLib and large artificial instances well exceeded our own expectations. We demonstrate that we are able to handle instances with up to a million nodes and several hundreds of terminals in 1.5 hours on a standard PC. On the rectilinear preprocessed instances from SteinLib, we observe a monotonous improvement for smaller values of ε, with an average gap below 1% for $\varepsilon = 0.1$. We compare our implementation against the well-known batched 1-Steiner heuristic and observe that on very large instances, we are able to produce comparable solutions much faster. We also present a thorough experimental evaluation of the influence of the various parameters of the PTAS and thus obtain a better understanding of their empirical effects.

Finally, in Chapter 4, we present a PTAS for the geometric Steiner tree problem with polygonal obstacles in the plane with running time $\mathcal{O}(n \log^2 n)$, where n denotes the number of terminals plus obstacle vertices. To this end, we show how a planar spanner of size $\mathcal{O}(n \log n)$ can be constructed that contains a $(1+\varepsilon)$-approximation of the optimal tree. Then one can find an approximately optimal Steiner tree in the spanner using the PTAS for Steiner tree in planar graphs [BKM09]. We prove this result for the Euclidean metric and also for all uniform orientation metrics, i.e. particularly the rectilinear and octilinear metrics.

Part II: Algorithms on Minor-Closed Graph Classes

In Chapter 5, we generalize the linear-time shortest-paths algorithm for planar graphs with nonnegative edge-weights of Henzinger, Klein, Rao, and Subramanian [HKRS97] to work for any proper minor-closed class of graphs. We argue that their algorithm cannot be adapted by standard methods to all proper minor-closed classes. By using recent deep results in graph minor theory by Reed and Wood [RW09], we show how to construct an appropriate recursive division in linear time for any graph excluding a fixed minor and how to transform the graph and its division afterwards, so that it has maximum degree three. Based on such a division, the original framework of Henzinger et al. can be applied. Afterwards, we show that using this algorithm, one can implement Mehlhorn's 2-approximation algorithm for the Steiner tree problem [Meh88] in linear time on these graph classes. This chapter is based on joint work with Matthias Müller-Hannemann [TM08, TM09b].

In Chapter 6, we improve the running time of the general algorithmic technique known as Baker's approach [Bak94] on H-minor-free graphs from $\mathcal{O}(n^{f(|H|)})$ to $\mathcal{O}(f(|H|)n^{\mathcal{O}(1)})$ showing that it is fixed-parameter tractable (FPT) with respect to the parameter $|H|$. The numerous applications include, for example, a 2-approximation for graph coloring, and PTASes for various problems such as dominating set and maximum cut, where we obtain similar improvements. The main tool that we use is an algorithmic decomposition theorem for H-minor-free graphs by Dawar, Grohe, and Kreutzer [DGK07] that can be applied in FPT-time with respect to parameter $|H|$.

On classes of odd-minor-free graphs, which have gained significant attention in recent years, we obtain a similar acceleration for computing a variant of the structural decomposition given by Demaine, Hajiaghayi, and Kawarabayashi [DHK10] and a Baker-style decomposition into two graphs of bounded treewidth. We use these algorithms to derive faster 2-approximations; furthermore, we present the first PTASes and subexponential FPT-algorithms for independent set and vertex cover on these graph classes using a novel dynamic programming technique.

We also introduce a technique to derive (nearly) subexponential parameterized algorithms on H-minor-free graphs. We provide a *uniform* algorithm running in time $\mathcal{O}(2^{\mathcal{O}_H(\sqrt{k \log n})} n^{\mathcal{O}(1)}) = \inf_{0<\varepsilon\leq 1} \mathcal{O}((1+\varepsilon)^k + n^{\mathcal{O}_H(1/\varepsilon)})$, where n is the size of the input and k is the number of vertices or edges in the solution. Our technique applies, in particular, to problems such as the general problem of finding a (directed) subgraph with a property, Steiner tree, (directed) longest path, and (connected/independent) dominating set, on some or all proper minor-closed graph classes, many of which were previously not even known to admit an algorithm with running time better than $\mathcal{O}(2^k n^{\mathcal{O}(1)})$. We obtain as a corollary that all problems with a minor-monotone subexponential kernel and amenable to our technique can be solved in subexponential FPT-time on H-minor free graphs.

Part III: Lower Bounds for the Complexity of Monadic Second-Order Logic

In Chapter 7, we develop several algorithmic tools – on one hand, for a number of (well-known) concepts from structural graph theory, and on the other hand, for some new structures that we introduce in our work. While these algorithms stand in their own right and are of independent interest, we will exploit them, in particular, to prove lower bounds for the tractability of monadic second-order logic (MSO_2) in Chapter 8. This part of the thesis is based on joint work with Stephan Kreutzer [KT10a, KT10b].

Specifically, in Chapter 7, we develop polynomial-time algorithms to compute the following structures:

(i) a *bramble* of the order of the square-root of the treewidth, up to logarithmic factors; brambles were introduced as the dual notion to treewidth and Grohe and Marx [GM09] recently showed that for polynomial-sized brambles, the order of the square-root of the treewidth is essentially the best we can hope for;

(ii) a *k-web*, a notion that we introduce based on the notion of k-meshes by Diestel, Gurbunov, Jensen, and Thomassen [DGJT99]; whereas k-meshes are not known to be constructable in polynomial time, we show that k-webs admit an efficient algorithm and can in turn be used to construct brambles, grid-like minors, and tree-ordered webs;

(iii) a *grid-like minor* of order polynomial in the treewidth; these structures were recently introduced by Reed and Wood [RW08] as a replacement for grid minors;

(iv) a *perfect bramble* of order polynomial in the treewidth; a bramble with a particularly simple structure that we introduce in this work and apply to obtain a *meta theorem* to decide certain subgraph-closed parameters in general graphs; we hope that this notion might have other interesting applications in the future;

(v) a *(labeled) tree-ordered web* of order polynomial in the treewidth; we define these objects by combining the concepts of k-webs and grid-like minors and introducing much more structure into it, so that it can be fully defined in MSO_2 and in fact, can be interpreted as a colored wall (or grid) in this logic; this will be the main algorithmic tool to derive our result about the complexity of MSO_2 in subgraph-closed classes of graphs in the following chapter.

Courcelle's famous theorem [Cou90] states that any property of graphs definable in MSO_2 can be decided in linear time on any class of graphs of bounded treewidth, or in other words, MSO_2-model checking is fixed-parameter tractable in linear time on any such class of graphs. From a logical perspective, Courcelle's theorem establishes a sufficient condition, or an upper bound, for tractability of MSO_2-model checking.

Whereas such upper bounds on the complexity of logics have received significant attention in the literature, almost nothing is known about corresponding lower bounds. In Chapter 8, we establish a strong lower bound for the complexity of monadic second-order logic. In particular, under a suitable complexity assumption, we show that if

(i) $\mathcal{C}$ is a class of colored graphs which is closed under recolorings, or

(ii) $\mathcal{C}$ is any class of graphs which is closed under taking subgraphs,

and additionally, the treewidth of $\mathcal{C}$ is not bounded polylogarithmically (in fact, $\log^c n$ for some small c suffices), then the model checking problem of MSO_2 is not fixed-parameter tractable on $\mathcal{C}$; and not even decidable in polynomial time if we allow the exponent of the polynomial to depend on the length of the given fixed formula.

Part I

Steiner Tree in Embedded Graphs: Extension, Engineering, and Application of a PTAS

1 Introduction to the Steiner Tree Problem and Its PTAS on Planar Graphs

In this chapter, we first introduce the Steiner tree problem and some of its variants and review some of the most important related literature. Afterwards, we briefly describe the PTAS for Steiner tree in planar graphs by Borradaile, Klein, and Mathieu [BKM09] as it builds the basis of the work we do in this part of the thesis.

1.1 The Steiner Tree Problem

We consider the following network design problem (see Figure 1.1):

Problem 1.1 (Steiner Tree)**.** Given an edge-weighted graph G and a set of terminals $R \subseteq V(G)$, find a tree in G of minimum weight that includes all the terminals.

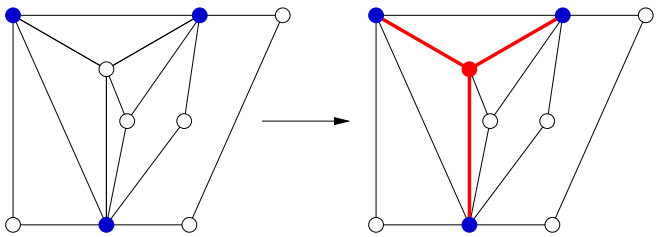


Figure 1.1: A graph with 3 blue terminals is given on the left; a Steiner tree with thick red lines is indicated on the right.

Steiner Tree is one of the most fundamental problems in computer science and serves as a testbed for many new algorithmic ideas – both in theory and in practice. It is one of the 21 problems that were first shown to be NP-hard by Karp [Kar72]. Later, Bern and Plassmann [BP89] showed that it is even APX-hard [BP89], i.e. does not admit a PTAS in general unless P = NP; in fact, it is already NP-hard to approximate Steiner Tree beyond a factor of 1.01053 [CC02]. Up until very recently, the best known approximation bound was $1.55 + \varepsilon$ as given by Robins and Zelikovsky [RZ00]; in a forthcoming paper, Byrka et al. [BGRS10] have announced a randomized approximation algorithm with expected approximation ratio of $1.39 + \varepsilon$. The problem is in FPT when parameterized by the number of terminals, as originally shown by Dreyfus and Wagner [DW72] and later improved by Erickson et al. [EMAFV87] and Bjorklund et al. [BHKK07]. As described in the next section, there is a well-known 2-approximation

algorithm for this problem [Cho78, Ple81] and Mehlhorn [Meh88] improved its running time to $\mathcal{O}(m + n \log n)$. Recently [TM09b], we further improved this algorithm, obtaining a linear running time on all proper minor-closed graph classes (which include planar graphs), see Chapter 5.

In planar graphs, the Steiner tree problem is also NP-hard [GJ77]. Erickson et al. [EMAFV87], Bern [Ber90], and Bern and Bienstock [BB91] showed that the problem is in FPT when parameterized by the genus and the number of faces or layers in which terminals appear. Very recently a PTAS has been presented by Borradaile, Klein, and Mathieu [BKM09]. See Section 1.5 for further details of this algorithm.

1.1.1 Geometric Variants

Problem 1.2 (ESMT)**.** Given a set of n terminals in the Euclidean plane, find the shortest tree that interconnects the given terminals.

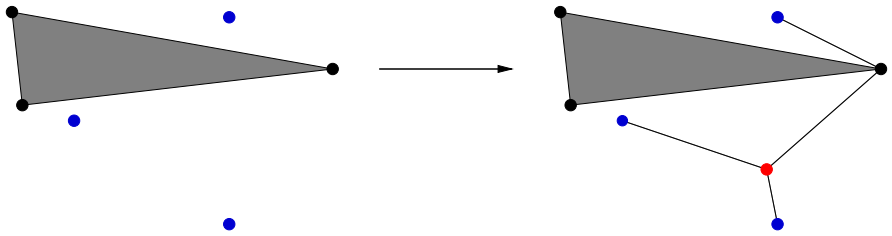


Figure 1.2: An example of ESMTO with 3 terminals and an obstacle with 3 corners; a solution is indicated on the right.

This problem is known as the *Euclidean Steiner minimum tree* (ESMT) problem. Note that the solution to an ESMT instance is a tree that may well contain additional points of the plane as vertices; in fact, this is what makes the problem difficult. In the *rectilinear* version (RSMT), we measure the length of edges in the Manhattan metric, i.e. the ℓ_1-metric. Both variants are well known to be NP-hard [GJ77, GGJ77]. A PTAS was presented by Arora [Aro98] and Mitchell [Mit99]. Rao and Smith [RS98] improved the running time of Arora's algorithm from $\mathcal{O}(n(\frac{1}{\varepsilon}\log n)^{\mathcal{O}(1/\varepsilon)})$ to $\mathcal{O}(2^{\text{poly}(1/\varepsilon)}n + n\log n)$ and this is the best running time known so far. The rectilinear case can be reduced to the planar graph version by using the so-called *Hanan-grid* [Han66, GC94], and the Euclidean case can be reduced to planar graphs as shown in [MT10] (see Chapter 4); the latter work actually shows that both variants can be reduced to planar graphs even in the presence of *obstacles* and thereby admit a PTAS in this case, too; in these versions, known as ESMTO and RSMTO, we are additionally given a set of polygonal obstacles whose interiors may not be crossed by the Steiner tree (see Figure 1.2).

1.1.2 Practical Significance

The Steiner tree problem and its many variations are also of high practical relevance; the applications reach from all kinds of network design to phylogenetic trees [FG82, HRW92, KR95, CKM+98, CD01]. Especially the geometric case with obstacles is very important in VLSI design since there are usually regions in the plane that may not be crossed by wire [GC94, ZW99]. Also, it is often only allowed to route the tree along a rectilinear or octilinear grid and so, Rsmto and similar variants are required [Tei02, CCK+03, KMZ03, PWZ04, MS06]

Since Steiner Tree has to be solved in many industry applications, numerous implementations exist that are able to solve this problem, often very well, in practice. The most important exact algorithms are due to Zachariasen and Winter [ZW99] for geometric instances, Koch and Martin [KM98] using integer linear programming techniques, and Polzin and Daneshmand [PD01, PD02, PD06] with the strongest results for general graphs. Also, many powerful heuristics exist, see, for example, [KR92, GRSZ94, PW02, KMZ03].

We refer to the books of Hwang et al. [HRW92] and Prömel and Steger [PS02] for further background on the Steiner tree problem.

1.2 A 2-Approximation Algorithm in $\mathcal{O}(n \log n + m)$ time

Let G be a given graph with a terminal set $R \subseteq V(G)$. The basic idea of the standard 2-approximation algorithm for Steiner Tree [Cho78, Ple81] is to consider the *distance network* N_D of the terminals: a complete graph that has one vertex for each terminal and in which the weight of every edge equals the shortest-path distance of the corresponding terminals in the input graph. Then a minimum spanning tree of the distance network is a 2-approximate Steiner tree of R in G.

Mehlhorn [Meh88] suggested an efficient implementation of this idea by observing that it is not necessary to construct the full distance network; instead, one can construct a *reduced* distance network $N_D^\star$ as follows. We first partition the graph into *Voronoi regions* with respect to the set of terminals. Every vertex of the graph belongs to the Voronoi region of its closest terminal (if a vertex happens to have the same distance to more than one terminal, it should belong to the Voronoi region of the terminal with the smallest index). Voronoi regions in graphs can be calculated easily using a shortest-paths computation: add a super-source s_0 to the graph and connect it to every terminal with a directed zero-weight edge; find the shortest paths from s_0 to every vertex and then remove s_0 from the resulting shortest-paths tree. The tree falls apart into $|R|$ connected components, each having a terminal as their root. These components correspond exactly to the Voronoi regions of the terminals. Using Dijkstra's algorithm [Dij59], one obtains a running time of $O(n \log n + m)$ for general graphs.

In the distance network $N_D^\star$, there exists an edge between two terminals u and v if and only if there exists an edge between two vertices x and y in G, so that x belongs to the Voronoi region of u and y belongs to the Voronoi region of v. The weight of such an edge is the length of the shortest such paths connecting u and v. Once the Voronoi

regions of G with respect to R are determined, $N_D^\star$ can be constructed in linear time using bucket sort.

After the distance network $N_D^\star$ is determined, one can find its minimum spanning tree and replace every edge with the corresponding shortest path in G. Mehlhorn shows that the resulting graph is indeed a tree and its weight is at most $(2 - \frac{2}{|R|})$ times the weight of the minimum Steiner tree of R in G. The implementation he offers runs in time $O(n \log n + m)$ for general graphs.

1.3 PTASes in Planar and Other Minor-Closed Graph Classes

Planar graphs constitute a very well-studied class of graphs in the literature. Even though most NP-hard problems remain NP-hard for their planar instances [GJ77, Lic82], they are often "easier" to handle; for example, many of them admit a PTAS. The research in this area was initiated already in 1979 by Lipton and Tarjan's planar separator theorem [LT79, LT80], where a PTAS for Independent Set on planar graphs was presented. An important milestone was Baker's approach [Bak94] that provided PTASes for a variety of problems, such as Vertex Cover, Independent Set, and Dominating Set on planar graphs. This approach was generalized to apex-minor-free graphs through the notion of bounded local treewidth by Eppstein [Epp00a] and to H-minor-free graphs by Grohe [Gro03]. Demaine and Hajiaghayi [DH05a] unified and generalized some of these results using the theory of bidimensionality. Still, for a number of problems PTASes are only known up to the class of apex-minor-free graphs, since these are exactly the minor-closed classes of graphs that have the bounded local treewidth property.

The traveling salesman problem (Tsp) - arguably one of the most prominent problems in computer science - was first shown to admit a PTAS on unweighted planar graphs [GKP95], then on weighted planar graphs [AGK$^+$98], weighted bounded-genus graphs [DHM07], and very recently unweighted apex-minor-free graphs [DHK09]. Since the Tsp requires a global connectivity measure, it can not be handled in the same way as bidimensional or similar local problems. The question whether it admits a PTAS on H-minor-free graphs is a very important open question in this area.

In [Kle08], Klein obtained a *linear time* approximation scheme for Tsp in weighted planar graphs and hence substantially accelerated the result of [AGK$^+$98]. In this work, he established a framework for obtaining PTASes in planar graphs that we are going to discuss in more detail in the next section. In [Kle06], Klein applied this framework to present a PTAS for Subset Tsp in planar graphs – a variant, where we are given a subset of the vertices of a planar graph and are asked for the shortest tour that visits this subset. Finally, Borradaile et al. [BKM07a] managed to extend these ideas and obtain a PTAS for Steiner Tree in planar graphs; shortly after, they improved the running time of their algorithm [BKM07b, BKM09]. Recently, this PTAS was generalized from planar graphs to bounded-genus graphs [BDT09] (see Chapter 2). Notice that almost a decade passed from the time a PTAS for planar Tsp was discovered until one for Steiner Tree was presented. The fact that in Steiner Tree we have to deal with a *subset* of the vertices makes it apparently substantially harder to approach than Tsp.

1.4 A Framework for PTASes in Planar Graphs

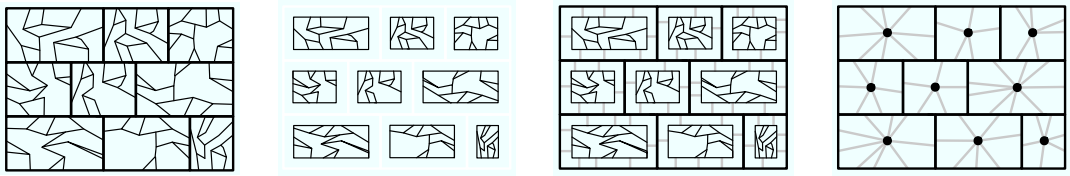
As mentioned above, in [Kle08], Klein proposed a framework to obtain a PTAS for a problem in planar graphs. We briefly review (a slightly modified version of) this framework in this section. In what follows, let G_{in} be the input graph and let OPT denote the weight of an optimal solution to a considered minimization problem; we also sometimes use the notation $\text{OPT}(G, R)$ to indicate the value of the optimal solution of a considered problem in a graph G on terminal set R.

1. **Spanner Step** The first step of the framework is to find a subgraph of the input graph that (i) has weight at most $\alpha \,\text{OPT}$, for a constant α; and (ii) still preserves an approximately optimal solution. Such a subgraph is (somewhat imprecisely) often called a *spanner*. Strictly speaking, a $(1+\varepsilon)$-*spanner* of a graph G is a spanning subgraph G' of G such that $\text{dist}_{G'}(u,v) \leq (1+\varepsilon)\,\text{dist}_G(u,v)$ for any two vertices $u, v \in V(G)$, i.e. a subgraph of G that approximately preserves distances. However, for a problem such as Steiner Tree this is not sufficient; we need to make sure that approximately optimal Steiner trees are preserved in the spanner. Rao and Smith [RS98] first obtained such a structure for Euclidean Steiner trees and called it a *banyan*. We use the general term spanner whenever no confusion arises. So, let G_{span} be the spanner obtained in this step from G_{in}.

2. **Thinning Step** In the second step, one finds a set of edges $S \subseteq E(G_{\text{span}})$ of weight at most $\mathcal{O}(\varepsilon \,\text{OPT})$ such that upon *contracting* S in G_{span}, one obtains a graph G_{thin} of *bounded treewidth*. In planar graphs, this can be achieved by applying Baker's decomposition [Bak94] to the *dual* of G_{span}: perform breadth-first-search (BFS) in the dual and, for a constant η, label the BFS-layers periodically by $0, \ldots, \eta - 1$. Now, contracting the edges with *any* fixed label in the primal graph results in a graph of bounded treewidth (recall that we identify the edges of the primal and the dual of a planar graph). Hence, by picking $\eta = \mathcal{O}(\frac{\alpha}{\varepsilon})$ and picking the label that induces the smallest total weight, we can guarantee to find a set S with the desired properties.

3. **Dynamic Programming Step** Since G_{thin} has bounded treewidth, we can usually calculate an optimal solution in G_{thin} in polynomial time using standard dynamic programming techniques (cf. [AP89, KS90]).

4. **Lifting Step** In the final step, we lift the solution of G_{thin} back to a solution of G_{in} by uncontracting the edges of S and adding them to the solution. This adds only an ε-fraction of OPT to the solution as the weight of S is bounded; furthermore, we only lost another ε-fraction in the spanner step and hence, by choosing the constants appropriately, we can guarantee that the final solution is near optimal.



(a) (b) (c) (d)

Figure 1.3: (a) An input graph G with mortar graph MG given by bold edges; (b) the set of bricks corresponding to MG; (c) the portal-connected graph, $\mathcal{B}^+(\mathrm{MG}, \theta)$; the portal edges are gray; (d) $\mathcal{B}^+(\mathrm{MG}, \theta)$ with the bricks contracted, resulting in the brick-contracted graph, $\mathcal{B}^\div(\mathrm{MG}, \theta)$; the dark vertices are brick vertices. These pictures are republished by courtesy of Glencora Borradaile [BKM09].

1.5 A PTAS for Steiner Tree in Planar Graphs

The main challenge in applying the framework above to Steiner Tree is the first step, i.e. obtaining a spanner that contains a $(1+\varepsilon)$-approximate solution and whose weight is bounded in terms of OPT. The first PTAS for this problem was obtained by Borradaile et al. [BKM07a] by showing how to accomplish this step in time $\mathcal{O}(2^{\mathrm{poly}(1/\varepsilon)} n \log n)$; this resulted in a PTAS with total running time $\mathcal{O}(2^{2^{\mathrm{poly}(1/\varepsilon)}} n \log n)$.[1] Shortly after, Borradaile et al. [BKM07b, BKM09] presented a modified algorithm in which they combined the spanner step and the dynamic programming step and obtained a PTAS with total running time *singly* exponential in $1/\varepsilon$, i.e. $\mathcal{O}(2^{\mathrm{poly}(1/\varepsilon)} n \log n)$. The centerpiece of their algorithm is the construction of a grid-like subgraph of the input graph, called a *mortar graph*. On one hand, the mortar graph can be used to construct a spanner and obtain a doubly exponential PTAS by the framework above; on the other hand, it can be directly incorporated into the dynamic programming to obtain a singly exponential algorithm. In what follows, we define the mortar graph and describe its construction along with the accelerated PTAS that it leads to.

1.5.1 The Mortar Graph and the Bricks

In [BKM09], the mortar graph is defined for planar graphs; but as this concept is generalized to graphs of bounded genus in [BDT09] (see Chapter 2), we define it here in this more general setting. A path P in a graph G is *ε-short in G* if for every pair of vertices x and y on P, the distance from x to y along P is at most $(1+\varepsilon)$ times the distance from x to y in G: $\mathrm{dist}_P(x,y) \leq (1+\varepsilon)\,\mathrm{dist}_G(x,y)$. Given a graph G embedded on a surface and a set of terminals R, a mortar graph is a subgraph of G with the following properties :

[1] The running time obtained in [BKM07a] is actually triply exponential in $1/\varepsilon$ but as pointed out in [BKM09], this can be improved to a doubly exponential algorithm by using Catalan structures.

Definition 1.3 (Mortar Graph and Bricks)**.** Given a graph G embedded on a surface of genus g, a set of terminals R, and a number $0 < \varepsilon \leq 1$, consider a subgraph $\mathrm{MG} := \mathrm{MG}(G, R, \varepsilon)$ of G spanning R such that each facial walk of MG encloses an area homeomorphic to an open disk. For each face F of MG, we construct a *brick* B of G by cutting G along the facial walk ∂F; B is the subgraph of G embedded inside the face, including ∂F. We denote this facial walk as the *mortar boundary* ∂B of B. We define the *interior* of B as B without the edges of ∂B. We call MG a *mortar graph* if for some constants $\alpha(\varepsilon, g)$ and $\kappa(\varepsilon, g)$ (to be defined later), we have $\ell(\mathrm{MG}) \leq \alpha\,\mathrm{OPT}$ and every brick B satisfies the following properties:

1. B is planar.
2. The boundary of B is the union of four paths in the clockwise order W, N, E, S.
3. Every terminal of R that is in B is on N or on S.
4. N is 0-short in B, and every proper subpath of S is ε-short in B.
5. There exists a number $k \leq \kappa$ and vertices $s_0, s_1, s_2, \ldots, s_k$ ordered from left to right along S such that, for any vertex x of $S[s_i, s_{i+1})$, the distance from x to s_i along S is less than ε times the distance from x to N in B: $\mathrm{dist}_S(x, s_i) < \varepsilon\ \mathrm{dist}_B(x, N)$.

The mortar graph and the set of bricks are illustrated in Figures 1.3 (a) and (b). Next we describe the algorithmic steps involved in constructing a mortar graph for a *planar* input graph G:

Decomposition into Strips The first step of the algorithm is to find a 2-approximate Steiner tree T. This can be done using our recent improvement on Mehlhorn's algorithm in linear time [TM09b, Meh88] (see Chapter 5). Afterwards, we cut open the tree along its Euler tour to obtain a distinguished face f_{out}; we may think of this face as the outer face of the graph. See Figure 1.4 (a), (b) and (c) for an illustration. Note that the weight of f_{out} is bounded by $4 \cdot \mathrm{OPT}$.

The goal of the strip decomposition is to decompose the graph into a number of components, called *strips*, so that the outer boundary of each strip consists of one shortest paths N, called the *north boundary* of the strip, and one ε-short path S, called the *south boundary* of the strip. We start with the whole graph and the boundary of its outer face, as described above; if there exists a subpath of the current outer boundary that is not ε-short, we determine a smallest subpath violating the condition, find a shortest path between its endpoints, and separate it from the current graph as a new strip (see Figure 1.4 (d)). Klein [Kle06] shows that the total weight of the strip boundaries is bounded by $4(1 + \varepsilon^{-1}) \cdot \mathrm{OPT}$. He describes an $\mathcal{O}(n \log n)$-time algorithm for the strip decomposition using dynamic trees (see, e.g. [TW05]) and his multiple-source shortest-paths algorithm [Kle05].

Adding Super-Columns The next step is to divide each strip into a number of bricks by adding so-called *super-columns*. The algorithm proceeds by finding a shortest path from every vertex on the south boundary to (any vertex on) the north boundary. This can be done by a single shortest-paths computation in $\mathcal{O}(n \log n)$ time [Kle06]. Starting

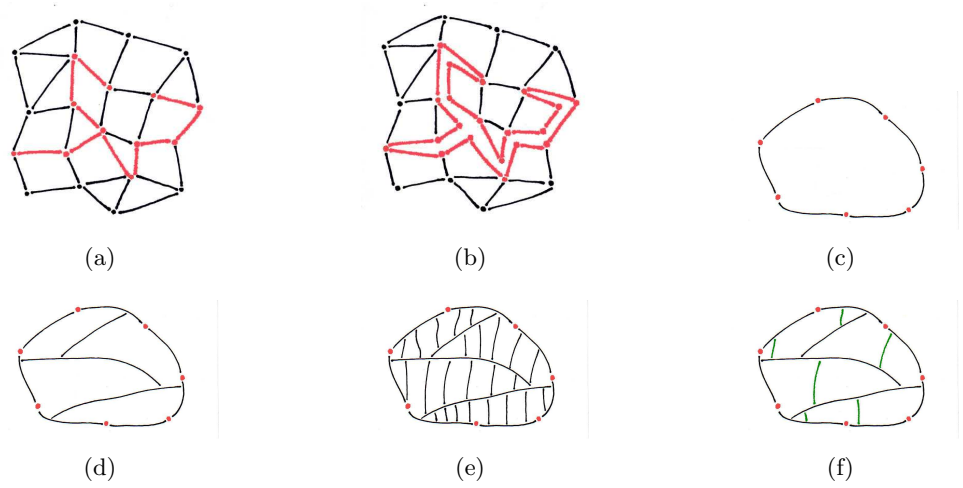


Figure 1.4: Construction of a mortar graph: (a) a 2-approximate Steiner tree (with red edges); (b) splitting the 2-approximation along its Euler tour; (c) thinking of the newly created face as the outer face; (d) adding shortest paths that comprise the strip-decomposition; (e) finding the columns inside each strip; (f) selecting the super-columns from among the columns. These pictures are republished by courtesy of Glencora Borradaile.

on the left-most vertex of the south boundary and moving to the right by one edge at a time, the algorithm extracts a set of *columns* as follows: if at any point the current sum of edges that are traversed so far is more than ε times the distance of the current node to the north boundary, this shortest path to the north boundary is added to the set of columns and the current sum of edges is reset to zero (see Figure 1.4 (e)). Next, for a constant κ and for each $i = 0, \ldots, \kappa - 1$, the columns with indices $i + c\kappa$, $c \in \mathbb{N}$, are considered, and the set with minimum weight is selected to be the set of super-columns in this strip (see Figure 1.4 (f)). This way, it is ensured that each brick contains at most κ columns, and furthermore, that the total weight of the super-columns is at most κ^{-1} times the total sum of all columns, which is in turn at most $4\kappa^{-1}\varepsilon^{-1}(1+\varepsilon^{-1}) \cdot \mathrm{OPT}$. We sometimes refer to κ as the *spacing of the super-columns* of the mortar graph.

The following results can be easily deduced from [Kle06] and [BKM09]:

Lemma 1.4 ([Kle06, BKM09])**.** *Let $0 < \varepsilon \leq 1$ and G be a planar graph with outer face f_{out} containing the terminals R and such that $\ell(f_{out}) \leq \alpha_0 \,\mathrm{OPT}$, for some constant α_0. For $\alpha = (2\alpha_0 + 1)\varepsilon^{-1}$, there is a mortar graph $\mathrm{MG}(G, R, \varepsilon)$ containing f_{out} whose length is at most $\alpha\,\mathrm{OPT}$ and whose super-columns have length at most $\varepsilon\,\mathrm{OPT}$ with $\kappa = \alpha_0\varepsilon^{-2}(1+\varepsilon^{-1})$. The mortar graph can be found in $O(n \log n)$ time.*

Theorem 1.5 ([Kle06, BKM09])**.** *Let G be a planar graph, R be a subset of vertices, and $0 < \varepsilon \leq 1$. For $\alpha = 9\varepsilon^{-1}$ there is a mortar graph $\mathrm{MG}(G, R, \varepsilon)$ of G such that the length of MG is $\leq \alpha\,\mathrm{OPT}$ and the super-columns of MG have length $\leq \varepsilon\,\mathrm{OPT}$ with $\kappa = 4\varepsilon^{-2}(1+\varepsilon^{-1})$. The mortar graph can be found in $\mathcal{O}(n \log n)$ time.*

1.5.2 Structure Theorem

Along with the mortar graph, Borradaile et al. [BKM09] define an operation $\mathcal{B}^+$ called *brick-copy* that allows a succinct statement of the Structure Theorem below. For each brick B, a subset of θ vertices is selected as *portals* such that the distance along ∂B between any vertex and the closest portal is at most $\ell(\partial B)/\theta$. For every brick B, embed B in the corresponding face of MG and connect every portal of B to the corresponding vertex of MG with a zero-length *portal edge*: this defines $\mathcal{B}^+(\mathrm{MG}, \theta)$ as illustrated in Figure 1.3 (d). We denote the set of all portal edges by E_{portal}. The following simple observation, proved in [BKM09], holds also for bounded-genus graphs:

Observation 1.6 ([BKM09])**.** *If A is a connected subgraph of $\mathcal{B}^+(\mathrm{MG}, \theta)$, then $A \setminus E_{portal}$ is a connected subgraph of G spanning the same vertices of G.*

The following Structure Theorem is the heart of the correctness of the PTAS.

Theorem 1.7 (Structure Theorem [BKM09])**.** *Let G be a graph embedded on a surface, $R \subseteq V(G)$ a given set of terminals, and $0 < \varepsilon \leq 1$. Let $\mathrm{MG}(G, R, \varepsilon)$ be a corresponding mortar graph of weight at most $\alpha\,\mathrm{OPT}$ and super-columns of weight at most $\varepsilon\,\mathrm{OPT}$ with spacing κ. There exist constants $\beta = o(\varepsilon^{-2.5}\kappa)$ and $\theta(\alpha, \beta)$ depending polynomially on α and β such that*

$$\mathrm{OPT}(\mathcal{B}^+(\mathrm{MG}, \theta), R) \leq (1 + c\varepsilon)\,\mathrm{OPT}(G, R)\,,$$

where c is an absolute constant.

The lengthy proof of this theorem is deeply involved and is presented in [BKM09] for planar graphs. Since the bricks are always planar, the proof for bounded-genus graphs follows as for the planar case [BDT09].

1.5.3 Obtaining a PTAS via a Spanner

As already mentioned, one way to obtain a PTAS for Steiner Tree in planar graphs is to construct a spanner and apply the framework of Klein [Kle08] reviewed in Section 1.4 above. Once a mortar graph MG is constructed, a spanner can be obtained as follows: for each brick B defined by MG and for each subset X of the portals of B, find the optimal Steiner tree for X in B (using the method of Erickson et al. [EMAFV87]); the spanner is the union of all these trees over all bricks plus the edges of the mortar graph. This results in the following theorem:

Theorem 1.8 ([BKM09])**.** *Let G be an edge-weighted planar graph and $R \subseteq V(G)$ a given set of terminals. There exists a spanner $G_{span} \subseteq G$ such that*

1. *G_{span} contains a $(1 + c\varepsilon)$-approximate solution to* Steiner Tree*; and*
2. *$\ell(G_{span}) \leq f(\varepsilon)\,\mathrm{OPT}$;*

where the function $f(\varepsilon)$ is singly exponential in a polynomial in ε^{-1} and c is an absolute constant. The spanner can be found in $O(n \log n)$ time.

Together with the framework of Section 1.4, Theorem 1.8 gives rise to a doubly exponential PTAS for Steiner Tree in planar graphs.

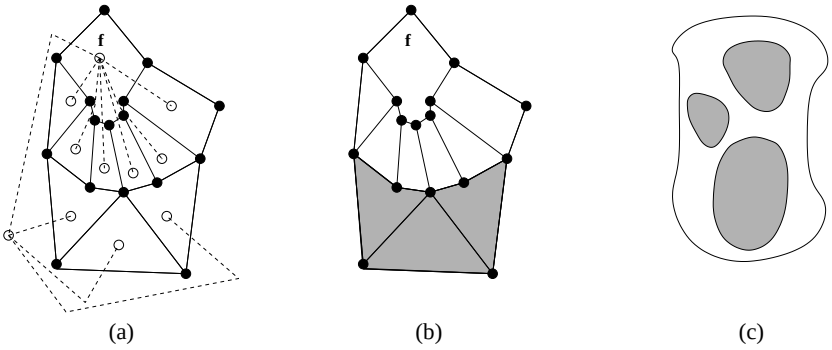


Figure 1.5: (a) A planar graph with black vertices and full lines and a breadth-first-search tree of the dual represented with white vertices and dashed lines; (b) a parcel decomposition corresponding to this BFS tree, rooted at face f, separating after level 1 of the tree; the white faces constitute the first parcel and the gray faces the second; (c) an illustration of a more complicated situation where many parcels are created at only one separating level.

1.5.4 Obtaining a Faster PTAS via Dynamic Programming over the Bricks

The idea of the faster PTAS is to apply the thinning step to the mortar graph instead of the spanner and then perform dynamic programming on the thinned mortar graph; but now, some leaves of the dynamic programming tree might actually be bricks in which case we have to compute and store all non-crossing Steiner trees of the brick using the algorithm of Erickson et al. [EMAFV87]. We review these steps in more detail below. A different thinning and dynamic programming technique that works more generally for graphs of bounded genus is discussed in Section 2.2.

Parcel Decomposition The thinning step is done somewhat differently in [BKM09] from the framework proposed by Klein [Kle08]. We obtain a set of edges S of weight at most $\frac{\varepsilon}{2}\,\mathrm{OPT}$ as described in Section 1.4 by performing breadth-first search in the dual of the mortar graph and considering the layers of the BFS-tree modulo a constant $\eta := 2\frac{\alpha}{\varepsilon} = 18\varepsilon^{-2}$. But instead of contracting these edges, we consider each region that is enclosed by S and call it a *parcel*; the set of edges S is called the set of *parcel boundaries*. Now each parcel has bounded dual radius – which implies bounded treewidth – and can be handled separately. See Figure 1.5 (a)–(b) for an illustration and note that there might be several parcels lying between the same two consecutive selected layers of the BFS-tree as shown in Figure 1.5 (c). In what follows, we set the weight of the parcel boundaries to zero as they can be added later to the final solution while causing only a small overhead of $\mathcal{O}(\varepsilon\,\mathrm{OPT})$.

Adding Auxiliary Terminals In order to ensure that the final solution is connected, we might have to add some auxiliary terminals to the parcels. Since every parcel P is a connected set of faces of the mortar graph, its outer boundary B_P is well-defined; let

R_P be the region of the plane enclosed by B_P and Q_P be the region of the plane outside B_P. If both R_P and Q_P contain a terminal, we have to add an auxiliary terminal to B_P to ensure connectivity of the final solution. As the weight of B_P is set to zero, i.e. B_P is completely added to the final solution, we may select an arbitrary vertex on B_P as such an additional terminal. In [BKM09] it is shown that this step can be performed in linear time.

Dynamic Programming in the Parcels The operation *brick contraction*, $\mathcal{B}^{\div}$, is defined as first applying $\mathcal{B}^{+}$ and then contracting each brick to become a single *brick vertex* of degree at most θ, where θ is the constant from the Structure Theorem 1.7. The graphs $\mathcal{B}^{+}(\mathrm{MG})$ and $\mathcal{B}^{\div}(\mathrm{MG})$ are illustrated in Figures 1.3 (d) and (e), respectively. Since each parcel P has dual radius at most η, we observe that $\mathcal{B}^{\div}(P)$ has dual radius at most $\theta\eta$. We consider a spanning tree of P, root it at some terminal, and add one portal edge per brick to it to obtain a spanning tree T of $\mathcal{B}^{\div}(P)$. Now each edge $e = vw$ of T defines a subproblem T_e of the dynamic program (DP) that includes w and all its descendants in T. The bounded dual radius property of P ensures that T_e is separated from the rest of the graph by a cut of at most ξ edges, where $\xi := 2\theta\eta + 1$ is a constant. By enumerating over all non-crossing partitions of this cut into sets that are to be connected inside the subproblem, one can fill the entries of the DP table – just as in standard DP algorithms on graphs of bounded treewidth [AP89, KS90]; however, for every leaf of T that is a contracted brick vertex, optimal Steiner trees for all non-crossing partitions of the portals inside the corresponding brick have to be calculated using the algorithm of Erickson et al. [EMAFV87]. A full description of the DP including implementation-level details is given in Section 3.2.3.

Putting Things Together In summary, the PTAS works as follows:

1. find the mortar graph MG;
2. decompose the graph into parcels;
3. set the weight of the parcel-boundaries to zero;
4. add auxiliary terminals if necessary;
5. for each parcel P, find the optimal Steiner tree in the brick-contracted graph $\mathcal{B}^{\div}(P)$ using dynamic programming;
6. return the union of the Steiner trees found in the parcels together with the parcel boundaries as the final solution (if necessary, prune some edges to obtain a tree).

The Structure Theorem 1.7 guarantees that the solution that we find for each parcel is within a factor of $(1 + c\varepsilon)$ of the optimum solution of the parcel; by running the algorithm with $\varepsilon' = \frac{\varepsilon}{2c}$, we can ensure that the approximation error incurred in this step is at most $\frac{\varepsilon}{2}\,\mathrm{OPT}$. We add the parcel boundaries to the solutions of the parcels to obtain one connected Steiner tree of the input graph; since the weight of the parcel boundaries is also bounded by $\frac{\varepsilon}{2}\,\mathrm{OPT}$, our final solution has weight at most $(1+\varepsilon)\,\mathrm{OPT}$, as desired.

Theorem 1.9 ([BKM09])**.** *There exists a* PTAS *for* Steiner Tree *in planar graphs with running time* $\mathcal{O}(2^{\mathrm{poly}(1/\varepsilon)} n \log n)$.

2 Extension of PTASes for Subset-Connectivity Problems from Planar to Bounded-Genus Graphs[1]

In many practical scenarios of network design, input graphs have a natural drawing on the sphere or equivalently the plane. In most cases, these embeddings have few crossings, either to avoid digging multiple levels of tunnels for fiber or cable or to avoid building overpasses in road networks. But a few crossings are common, and can easily come in bunches where one tunnel or overpass might carry several links or roads. Thus we naturally arrive at graphs of small (bounded) genus, which is the topic of this chapter.

We develop a PTAS framework for subset-connectivity problems on edge-weighted graphs of bounded genus. In general, we are given a subset of the nodes, called *terminals*, and the goal is to connect the terminals together with some substructure of the graph by using cost within $1+\varepsilon$ of the minimum possible cost. Our framework applies, in particular, to the well studied Steiner Tree problem, where we require the substructure to be connected. This problem will be the main focus of this chapter. Our framework yields the first PTAS for this problem in bounded-genus graphs. The PTAS is efficient, running in $\mathcal{O}(f(\varepsilon, g)\, n + h(g)\, n \log n) = \mathcal{O}_{\varepsilon,g}(n \log n)$ time for graphs embedded on orientable surfaces and nonorientable surfaces (we usually omit the mention of $f(\varepsilon, g)$ and $h(g)$ by assuming ε and g are constant, but we later bound $f(\varepsilon, g)$ as singly exponential in a polynomial in $1/\varepsilon$ and g and $h(g)$ as singly exponential in g). In contrast, we know that Steiner Tree is APX-complete (and constant-factor-approximable) for general graphs.

We build upon the recent PTAS framework of Borradaile, Klein, and Mathieu [BKM09] for subset-connectivity problems on planar graphs. In contrast to the planar-graph framework, our PTASes have the attractive feature that they run correctly on all graphs with the performance degrading with the genus. Also, our result is strictly more general: any problem to which the previous planar-graph framework applies readily generalizes to our framework as well, resulting in a PTAS for bounded-genus graphs.

For example, the Subset Tsp problem, where we are looking for a tour that passes through a given set of terminals, has been shown to admit a PTAS in planar graphs by Klein [Kle06]. We can directly combine the techniques of [Kle06] with our framework to generalize this PTAS to bounded-genus graphs.[2] As another example, Borradaile

[1] This chapter is based on joint work with Glencora Borradaile and Erik Demaine [BDT09].

[2] In fact, we show in [BDT09] how to unify the approaches of [Kle06] and [BKM09] and obtain a structure theorem for Subset Tsp, so that the mortar graph framework can be applied to it as well. However, as this part of the paper is due to Glencora Borradaile, we choose not to include it here.

and Klein [BK08b] have recently claimed a PTAS for the $\{0,1,2\}$-edge-connectivity Survivable Network problem using the planar framework. This will imply a similar result in bounded-genus graphs. In this problem, the substructure we are after must have $\min\{c_x, c_y\}$ edge-disjoint paths connecting terminals x and y, where each $c_x \in \{0,1,2\}$; we allow the substructure to include multiple copies of an edge in the graph, but pay for each copy. In particular, if $c_x = 1$ for all terminals x and y, then we obtain the Steiner tree problem; if $c_x = 2$ for all terminals x and y, then we obtain the *minimum-cost* 2*-edge-connected submultigraph* problem.

Our techniques for attacking bounded-genus graphs include two recent results: decompositions into bounded-treewidth graphs via contractions [DHM07] and fast algorithms for finding the shortest noncontractible cycle [CC07]. We also use a simplified version of an algorithm for finding a short sequence of loops on a topological surface [EW05], and sophisticated dynamic programming. Our aim is to prove the following theorem:

Theorem 2.1. *There exists a* PTAS *for* Steiner Tree *in edge-weighted graphs of genus g with running time $\mathcal{O}(2^{\mathrm{poly}(\varepsilon^{-1},g)}n + 2^{\mathrm{poly}(g)}n\log n)$.*

Notation and Basic Definitions

We use the notions about graphs embedded on surfaces as introduced in Section A.2 based on the book of Mohar and Thomassen [MT01]. In this chapter, all graphs $G = (V, E)$ have n vertices, m edges, and are undirected with edge lengths (weights). The length of an edge e, subgraph H, and set of subgraphs $\mathcal{H}$ are denoted by $\ell(e)$, $\ell(H)$, and $\ell(\mathcal{H})$, respectively. The shortest distance between vertices x and y in a graph G is denoted $\mathrm{dist}_G(x,y)$. Furthermore, ∂G denotes the boundary of a graph G embedded in the plane or on a disc.

We consider 2-cell embedded graphs with a combinatorial embedding on an orientable or nonorientable surface. We let f denote the number of faces of G and $g = 2+m-n-f$ the Euler genus. Recall that the dual $G^\star$ of G is defined on the same set of edges as G.

A cycle of an embedded graph is *contractible* if it can be continuously deformed to a point; otherwise it is *noncontractible*. The operation of *cutting along a twosided cycle* C is essentially: partition the edges adjacent to C into left and right edges and replace C with two copies C_ℓ and C_r, adjacent to the left or right edges, accordingly. The inside of these new cycles is "patched" with two new faces. If the resulting graph is disconnected, the cycle is called *separating*, otherwise *nonseparating*. Cutting along a onesided cycle C on nonorientable surfaces is defined similarly, only that C is replaced by one bigger cycle C' that contains every edge of C exactly twice. See [MT01, pages 105–106] for further technical details.

The input graph is $G_0 = (V_0, E_0)$ and has genus g_0; the terminal set is Q. We assume G_0 is equipped with a combinatorial embedding; such an embedding can be found in linear time if the genus is known to be fixed, see [Moh99]. Let $\mathcal{P}$ be the considered subset-connectivity problem. In Section 2.1.1, we show how to find a subgraph $G = (V, E)$ of G_0 such that for $0 \le \varepsilon \le 1$ any $(1+\varepsilon)$-approximate solution of $\mathcal{P}$ in G_0 also exists in G. Hence, we may use G instead of G_0 in the rest of the paper. Note that as a subgraph of G_0, G is automatically equipped with a combinatorial embedding.

Let OPT denote the length of an optimal Steiner tree spanning terminals Q. We define $\mathrm{OPT}_{\mathcal{P}}$ to be the length of an optimal solution to problem $\mathcal{P}$. For the problems that we solve, we require that $\mathrm{OPT}_{\mathcal{P}} = \Theta(\mathrm{OPT})$ and in particular that $\mathrm{OPT} \leq \mathrm{OPT}_{\mathcal{P}} \leq \mu\,\mathrm{OPT}$. The constant μ will be used in Section 2.1 and is equal to 1 for the Steiner tree problem. We choose this presentation to easily allow for generalizations to other problems. For example, for both Subset Tsp and $\{0,1,2\}$-edge-connectivity Survivable Network, we have $\mu = 2$. This requirement is also needed for the planar case; see [BK08b]. Because $\mathrm{OPT} \leq \mathrm{OPT}_{\mathcal{P}}$, upper bounds in terms of OPT hold for all the problems herein. As a result, we can safely drop the $\mathcal{P}$ subscript henceforth.

We show how to obtain a $(1+c\varepsilon)\,\mathrm{OPT}$ solution for an absolute constant c. To obtain a $(1+\varepsilon)\,\mathrm{OPT}$ solution, we can simply use $\varepsilon' = \varepsilon/c$ as input to the algorithm. We follow the framework of Borradaile et al. [BKM09] as introduced in Section 1.5: first, we construct a mortar graph MG and then perform dynamic programming on the mortar graph and the bricks. We also show how to obtain a spanner and apply Klein's framework [Kle08] as reviewed in Section 1.4.

2.1 Mortar Graph and Structure Theorems

Let $G_0 = (V_0, E_0)$ be the input graph of genus g_0 and Q be the terminal set. In a first preprocessing step, we delete a number of unnecessary vertices and edges of G_0 to obtain a graph $G = (V, E)$ of genus $g \leq g_0$ that still contains every $(1+\varepsilon)$-approximate solution for terminal set Q for all $0 \leq \varepsilon \leq 1$ while fulfilling certain bounds on the length of shortest paths. In the next step, we find a *cut graph* CG of G that contains all terminals and whose length is bounded by a constant times OPT. We cut the graph open along CG, so that it becomes a planar graph with a simple cycle σ as boundary, where the length of σ is twice that of CG. See Figure 2.1 for an illustration. Afterwards, the remaining steps of building the mortar graph can be the same as in the planar case, by way of Theorem 1.4.

For an edge $e = vw$ in G_0, we let $\mathrm{dist}_{G_0}(r, e) = \min\{\mathrm{dist}_{G_0}(r, v), \mathrm{dist}_{G_0}(r, w)\} + \ell(e)$ and say that e is *at distance* $\mathrm{dist}_{G_0}(r, e)$ from r. If the root vertex represents a contracted graph H, we use the same terminology with respect to H.

2.1.1 Preprocessing the Input Graph

Our first step is to apply the following preprocessing procedure:

Algorithm Preprocess(G_0, Q, μ).
Input. an arbitrary graph G_0, terminals $Q \subseteq V(G_0)$, a constant μ
Output. a preprocessed subgraph of G_0

1. Find a 2-approximate Steiner tree T_0 for Q in G_0; contract T_0 to a vertex r.
2. Find a shortest-path tree rooted at r.
3. Delete all vertices v and edges e of G_0 with $\mathrm{dist}_{G_0}(r, v), \mathrm{dist}_{G_0}(r, e) > 2\mu\ell(T_0)$.

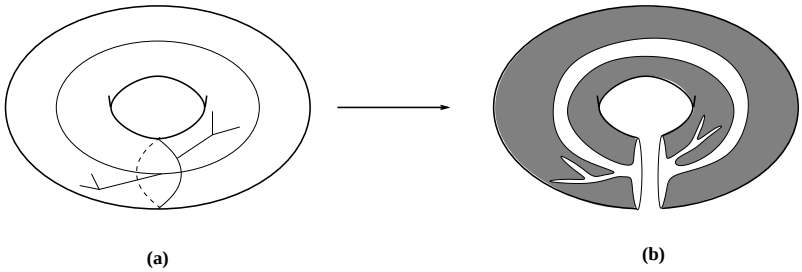


Figure 2.1: (a) a cut graph of a tree drawn on a torus; (b) the result of cutting the surface open along the cut graph: the shaded area is homeomorphic to a disc and the white area is the additional face of the planarized surface.

Any deleted vertex or edge is at distance $> 2\mu\ell(T_0) > 2\mu\,\mathrm{OPT}$ from any terminal and hence can not be part of a $(1+\varepsilon)$-approximation for any $0 \leq \varepsilon \leq 1$. We call the resulting graph $G = (V, E)$ and henceforth use G instead of G_0 in our algorithm. The preprocessing step can be accomplished in linear time: T_0 can be calculated with our recent improvement on Mehlhorn's algorithm [Meh88, TM09b] (see Chapter 5), and the shortest path tree with Henzinger et al.'s algorithm [HKRS97]. Trivially, we have

Proposition 2.2. *All vertices and edges of G are at distance at most $4\mu\,\mathrm{OPT}$ from T_0.*

2.1.2 Constructing the Cut Graph

A central fact that we use in this section and also in other parts of our work is the following observation, whose proof is folklore; see e.g. [Epp03] and cf. Figure 1.5.

Observation 2.3 (folklore)**.** *Let G be a planar graph and T a spanning tree of G. Then the set of edges $E(G) - E(T)$ induces a spanning tree $T^\star$ in the dual $G^\star$. If T is a minimum spanning tree of G, then $T^\star$ is a maximum spanning tree of $G^\star$.*

A similar lemma also holds for bounded-genus graphs: if T is a (minimum) spanning tree of G and $T^\star$ a (maximum) spanning tree of $G^\star - E(T)$, then $T^\star$ is a (maximum) spanning tree of $G^\star$ and the size of the set of remaining edges $X := E(G) - E(T) - E(T^\star)$ is g, the Euler genus of G, by Euler's formula. Eppstein [Epp03] defines such a triple $(T, T^\star, X)$ as a *tree-cotree decomposition* of G and shows that such a decomposition can be found in linear time for graphs on both orientable and nonorientable surfaces.

In order to construct a cut graph, we start again with a 2-approximation T_0 and contract it to a vertex r. Next, we look for a *system of loops* rooted at r: iteratively find short nonseparating cycles through r and cut the graph open along each cycle. Erickson and Whittlesey [EW05] showed that, for orientable surfaces, taking the *shortest* applicable cycle at each step results in the shortest system of loops through r. They suggest an implementation of their algorithm using the tree-cotree decomposition $(T, T^\star, X)$ of Eppstein [Epp03] in linear time. Eppstein showed that the set of elementary cycles $\{\mathrm{loop}(T, e) : e \in X\}$ is a cut graph of G where $\mathrm{loop}(T, e)$ is the closed walk formed by the paths in T from r to the endpoints of e plus the edge e. Eppstein's decomposition

also works for nonorientable embeddings. As we only need to bound the length of our cut graph, we present a simpler algorithm below:

Algorithm PLANARIZE(G_0, Q, μ)**.**
Input. a graph G_0 of fixed genus g, terminals $Q \subseteq V(G_0)$, a constant μ
Output. a preprocessed subgraph $G \subseteq G_0$ and a cutgraph CG of G

1. Apply PREPROCESS(G_0, Q, μ) and let G be the obtained subgraph.
2. Find a 2-approximate Steiner tree T_0 for Q in G; contract T_0 to a vertex r.
3. Find a shortest paths tree SPT rooted at r.
4. Uncontract r and set $T_1 = T_0 \cup \text{SPT}$. ($T_1$ is a spanning tree of G)
5. Find a spanning tree $T_1^\star$ in $G^\star - E(T_1)$. ($T_1^\star$ is a spanning tree of $G^\star$)
6. Let $X := E(G) - E(T) - E(T^\star)$.
7. Return CG $:= T_0 \cup \{\text{loop}(T_1, e) : e \in X\}$ together with G.

Lemma 2.4. *The algorithm* PLANARIZE *returns a cut graph* CG *such that cutting G open along* CG *results in a planar graph G_p with a face f_σ whose facial walk σ*

(i) is a simple cycle;
(ii) contains all terminals (some terminals might appear more than once as multiple copies might be created during the cutting process); and
(iii) has length $\ell(\sigma) \leq 2(8\mu g + 2)\,\text{OPT}$.

The algorithm can be implemented in linear time.

Proof. Clearly, $(T_1, T_1^\star, X)$ is tree-cotree decomposition of G and so, by Eppstein's Lemma [Epp03], CG is a cut graph. By Euler's formula, we get that $|X| = g$, the Euler genus of G. Each edge $e = vw \in X$ completes a (nonsurface-separating, not necessarily simple) closed walk as follows: a shortest path P_1 from T_0 to v, the edge e, a shortest path P_2 from w to T_0 and possibly a path P_3 in T_0. By Proposition 2.2, we know that e is at distance at most $4\mu\,\text{OPT}$ from T_0 and so, P_1, P_2, and at least one of $P_1 \cup \{e\}$ and $P_2 \cup \{e\}$ have length at most $4\mu\,\text{OPT}$. Hence, we have that $\ell(P_1 \cup \{e\} \cup P_2) \leq 8\mu\,\text{OPT}$. Because there are (exactly) g such cycles in CG, we get that

$$\ell(\text{CG}) \leq g \cdot 8\mu\,\text{OPT} + \ell(T_0) \leq (8\mu g + 2)\,\text{OPT}\,.$$

Because CG is a cut graph, it follows that it consists of a single facial walk σ'; this follows easily from Euler's formula and the fact that CG has Euler genus g with some k vertices and $k+g-1$ edges. So, σ' contains every edge of CG exactly twice (cf. [MT01]), i.e. $\ell(\sigma') = 2\ell(\text{CG})$. Cutting the graph open along σ' results in a planar graph with a simple cycle $\sigma = \sigma'$ as its boundary, as desired (see Fig. 2.1).

As mentioned in the previous section, T_0 and SPT can be computed in linear time on bounded-genus graphs [HKRS97, TM09b]. $T_1^\star$ can be obtained, for example, by a simple breadth-first-search in the dual. The remaining steps can also easily be implemented in linear time. □

2.1.3 Constructing the Mortar Graph

Theorem 2.5. *Let an embedded edge-weighted graph G of Euler genus g, a subset of its vertices Q, an $0 < \varepsilon \leq 1$, and $\mu \geq 1$ be given. For $\alpha = (32\mu g + 9)\varepsilon^{-1}$, there is a mortar graph $\mathrm{MG}(G, Q, \varepsilon)$ of G such that the length of MG is $\leq \alpha\,\mathrm{OPT}$ and the super-columns of MG have length $\leq \varepsilon\,\mathrm{OPT}$ with spacing $\kappa = (16\mu g + 4)\varepsilon^{-2}(1+\varepsilon^{-1})$. The mortar graph can be found in $\mathcal{O}(n \log n)$ time.*

Proof. Let G_p be the result of planarizing G as guaranteed by Lemma 2.4. G_p is a planar graph with boundary σ such that σ spans Q and has length $\leq 2(8\mu g + 2)\,\mathrm{OPT}$. Let MG be the mortar graph guaranteed by Theorem 1.4 as applied to G with σ as its outer face. Every edge of MG corresponds to an edge of G. Let MG' be the subgraph of G composed of edges corresponding to MG. Every face f of MG (other than σ) corresponds to a face f' of MG' and the interior of f' is homeomorphic to a disk on the surface in which G is embedded. It is easy to verify that MG' is indeed a mortar graph of G; and the length bounds specified in the statement of the theorem follow directly from Theorem 1.4 and the bound on the length of σ. □

2.1.4 Structure Theorems for Subset-Connectivity Problems

We stated the Structure Theorem 1.7 for Steiner Tree already in its full generality for bounded-genus graphs. Note that it is due to the special way of constructing our mortar graph that we can obtain this theorem immediately from the planar case that was proven by Borradaile et al. [BKM09]: the crucial point here is that our bricks are always planar – even when the given graph is embedded in a surface of higher genus.

Recall the definition of the graph $\mathcal{B}^+(\mathrm{MG}, \theta)$ as given in Section 1.5.2 and Figure 1.3 (d). The Structure Theorem essentially says that there is a constant θ depending polynomially on ε^{-1} such that in finding a near-optimal solution to G, we can restrict our attention to $\mathcal{B}^+(\mathrm{MG}, \theta)$. Whenever we wish to apply our framework to a new problem, it is essential to prove a similar structure theorem for the considered problem. This is exactly what we did for Subset Tsp in [BDT09] and Borradaile and Klein [BK08b] did for Survivable Network. We refer to the cited papers for further details on these problems. Note also that for Subset Tsp, it is possible to obtain a singly exponential algorithm by following the spanner construction of Klein [Kle06] after performing the planarizing step (Lemma 2.4).

2.2 A PTAS for Steiner Tree in Bounded-Genus Graphs

As in Chapter 1, we present two methods to obtain a PTAS for Steiner Tree. One is based on finding a spanner and applying the framework of Klein [Kle08] as reviewed in Section 1.4; the other one is based on dynamic programming directly on the mortar graph and the bricks. While both methods result in $\mathcal{O}(n \log n)$ algorithms, the first method is doubly exponential in a polynomial in g and ε^{-1} and the second is singly exponential.

2.2.1 Spanner Construction

Recall that a spanner is a subgraph of length $\mathcal{O}_{\varepsilon,g}(\mathrm{OPT})$ that contains a $(1+\varepsilon)$-approximate solution. Here we show how to find a spanner for Steiner Tree in bounded-genus graphs. After a mortar graph is computed, the construction is, in fact, exactly the same as in Section 1.5.3, namely:

> For each brick B defined by MG and for each subset X of the portals of B, find the optimal Steiner tree of X in B (using the method of Erickson et al. [EMAFV87]). The spanner G_{span} is the union of all these trees over all bricks plus the edges of the mortar graph.

Theorem 2.6 (Spanner Theorem)**.** *Let G be an edge-weighted graph embedded on a surface of Euler genus g and $Q \subseteq V(G)$ a given set of terminals. There exists a spanner $G_{span} \subseteq G$ such that*

1. *G_{span} contains a $(1+c\varepsilon)$-approximate Steiner tree; and*
2. *$\ell(G_{span}) \leq f(\varepsilon, g)\,\mathrm{OPT}$;*

where the function $f(\varepsilon, g)$ is singly exponential in a polynomial in ε^{-1} and g, and c is an absolute constant. The spanner can be found in $\mathcal{O}(n \log n)$ time.

Proof. Given a mortar graph $\mathrm{MG}(G, Q, \varepsilon)$ as guaranteed by Theorem 2.5, a spanner is constructed as specified above. As in [BKM09], the time to find G_{span} is $\mathcal{O}(n \log n)$. It was proved in [BKM09] that $\ell(G_{\mathrm{span}}) \leq (1+2^{\theta+1})\ell(\mathrm{MG})$. Therefore, $\ell(G_{\mathrm{span}}) \leq (1+2^{\theta+1})\alpha\,\mathrm{OPT}$ and $f(\varepsilon, g) = (1+2^{\theta+1})\alpha$ (recall that α and θ depend polynomially on ε^{-1} and g). It remains to show that G_{span} contains a near-optimal solution; but this follows directly from the Structure Theorem 1.7. □

2.2.2 PTAS via Spanner

In order to apply the PTAS framework of Klein [Kle08] to bounded-genus graphs, we need the following Contraction Decomposition Theorem due to Demaine et al.:

Theorem 2.7 ([DHM07, Theorem 1.1])**.** *For a fixed genus g, and any integer $\eta \geq 2$ and for every graph G of Euler genus at most g, the edges of G can be partitioned into η sets such that contracting any one of the sets results in a graph of treewidth at most $\mathcal{O}(g^2 \cdot \eta)$. Furthermore, such a partition can be found in $\mathcal{O}(g^{5/2} n^{3/2} \log n)$ time.*

Recent techniques [CC07] for finding shortest noncontractible cycles of embedded graphs have improved the above running time to $\mathcal{O}(2^{\mathrm{poly}(g)} n \log n)$.

We review the four steps of the framework in our setting:

1. **Spanner Step** Find a spanner G_{span} of G according to Theorem 2.6.

2. **Thinning Step** For $\eta = f(\varepsilon, g)/\varepsilon$ (where $f(\varepsilon, g)$ is the function given in Theorem 2.6), let $S_1, \ldots, S_\eta$ be the partition of the edges of G_{span} as guaranteed by Theorem 2.7. Let S^* be the set in the partition with minimum weight: $\ell(S^*) \leq \varepsilon\,\mathrm{OPT}$. Let G_{thin} be the graph obtained from G_{span} by contracting the edges of S^*. By Theorem 2.7, G_{thin} has treewidth at most $\mathcal{O}(g^2 \varepsilon^{-1} f(\varepsilon, g))$.

3. **Dynamic Programming Step** Use dynamic programming (see, e.g. [KS90]) to find the optimal solution to the problem in G_{thin}.

4. **Lifting Step** Convert this solution to a solution in G by incorporating some of the edges of S^*; see, e.g., [Taz06, BKM09].

Analysis of the running time. By Theorem 2.6, the spanner step takes $\mathcal{O}_{\varepsilon,g}(n \log n)$ time (with singly exponential dependence on polynomials in g and ε^{-1}). By Theorem 2.7, thinning takes time $\mathcal{O}_g(n \log n)$ using [CC07]. Dynamic programming takes time $2^{\mathcal{O}(g^2\varepsilon^{-1}f(\varepsilon,g))}n$: because $f(\varepsilon, g)$ is singly exponential in polynomials in g and ε^{-1}, this step is doubly exponential in polynomials in g and ε^{-1}. Lifting takes linear time. Hence, the overall running time is $\mathcal{O}(2^{\mathcal{O}(g^2\varepsilon^{-1}f(\varepsilon,g))}n + 2^{\text{poly}(g)}n \log n)$.

2.2.3 PTAS via Dynamic Programming over the Bricks

We proceed in a similar fashion as in the case of planar graphs [BKM09] as reviewed in Section 1.5.4. However, as the concept of bounded dual radius does not apply straightforwardly to bounded-genus graphs, we have to deal with treewidth directly. We do this by applying the Contraction Decomposition Theorem 2.7 of Demaine et al. [DHM07] and a generalized dynamic programming technique. Recall that the operation *brick-contraction* $\mathcal{B}^{\div}$ is defined as the application of the operation $\mathcal{B}^{+}$ followed by contracting each brick to become a single vertex of degree at most θ (see Figure 1.3 (e)).

Algorithm THINNING(G, MG).
Input. a graph G of fixed genus g, a mortar graph MG of G
Output. a set $S^\star \subseteq E(\mathcal{B}^{\div}(\text{MG}))$, a tree decomposition (T, χ) of $\mathcal{B}^{\div}(MG)/S^\star$

1. Assign the weight $\ell(\partial F)$ to each portal edge e enclosed in a face F of $\mathcal{B}^{\div}(\text{MG})$.
2. Apply the Contraction Decomposition Theorem 2.7 to $\mathcal{B}^{\div}(\text{MG})$ with $\eta := 3\theta\alpha\varepsilon^{-1}$ to obtain edge sets $S_1, \ldots, S_\eta$; let $S^\star$ be the set of minimum weight.
3. If $S^\star$ includes a portal edge e of a brick B enclosed in a face F of MG, add ∂F to $S^\star$ and mark B as ignored.
4. Let $\text{MG}_{\text{thin}} := \mathcal{B}^{\div}(\text{MG})/S^\star$ (but without deleting parallel portal edges).
5. Let (T, χ) be a tree decomposition of width $\mathcal{O}(g^2 \cdot \eta)$ of MG_{thin}.
6. For each vertex b of MG_{thin} that represents an unignored contracted brick with portals $\{p_1, \ldots, p_\theta\}$:
   6.1. Replace every occurrence of b in χ with $\{p_1, \ldots, p_\theta\}$;
   6.2. Add a bag $\{b, p_1, \ldots, p_\theta\}$ to χ and connect it to a bag containing $\{p_1, \ldots, p_\theta\}$.
7. Reset the weight of the portal edges back to zero.
8. Return (T, χ) and $S^\star$.

We apply the Contraction Decomposition Theorem to $\mathcal{B}^{\div}(\text{MG})$ and contract a set of edges $S^\star$ in $\mathcal{B}^{\div}(\text{MG})$. However, we apply a special weight to portal edges so as to prevent

them from being included in $S^\star$. Also, in $\mathcal{B}^\div(\mathrm{MG})$, we slightly modify the definition of contraction: after contracting an edge, we do not delete parallel portal edges. Because portal edges connect the mortar graph to the bricks, they are not parallel in the graph in which we find a solution via dynamic programming.

Lemma 2.8. *The algorithm* Thinning(G, MG) *returns a set of edges* $S^\star$ *and a tree decomposition* (T, χ) *of* $\mathcal{B}^\div(\mathrm{MG})/S^\star$*, so that*

(i) the treewidth of (T, χ) *is at most* ξ *where* $\xi(\varepsilon, g) = \mathcal{O}(g^2\eta\theta) = \mathcal{O}(g^3\varepsilon^{-2}\theta^2)$*; in particular,* ξ *is polynomial in* ε^{-1} *and* g*;*
(ii) every brick is either marked as ignored or none of its portal edges are in $S^\star$*; and*
(iii) $\ell(S^\star) \le \varepsilon\,\mathrm{OPT}$.

Proof. We first verify that (T, χ) is indeed a tree decomposition. For a vertex v and a tree decomposition (T', χ'), let T'_v denote the subtree of T' that contains v in all of its bags. Let us denote the tree decomposition of step (5) by (T^0, χ^0). For each brick vertex b and each of its portals p_i, we know that T^0_b is connected and $T^0_{p_i}$ is connected and that these two subtrees intersect; it follows that after the replacement in step (6.2), we have that $T_{p_i} = T^0_b \cup T^0_{p_i}$ is a connected subtree of T and hence, (T, χ) is a correct tree decomposition. Note that Theorem 2.7 guarantees a tree decomposition of width $\mathcal{O}(g^2\eta)$ if any of $S_1, \dots, S_\eta$ are contracted; and in step (3), we only add to the set of edges to be contracted. Hence, the treewidth of (T^0, χ^0) is indeed $\mathcal{O}(g^2\eta)$ and with the construction in line (6.1), the size of each bag will be multiplied by a factor of at most θ. This shows the correctness of claim (i). The correctness of claim (ii) is immediate from the construction in line (3). It remains to verify claim (iii).

Let L denote the weight of $\mathcal{B}^\div(\mathrm{MG})$ after setting the weights of the portal edges according to step (1) of the algorithm. We have that

$$L \le \ell(\mathrm{MG}) + \sum_F \ell(\partial F)\theta \le \alpha\,\mathrm{OPT} + \theta \sum_F \ell(\partial F) \le \alpha\,\mathrm{OPT} + \theta \cdot 2\alpha\,\mathrm{OPT} \le 3\theta\alpha\,\mathrm{OPT}\ .$$

Hence, the weight of $S^\star$, as selected in step (2), is at most $L/\eta \le \frac{3\theta\alpha\,\mathrm{OPT}}{3\theta\alpha\varepsilon^{-1}} \le \varepsilon\,\mathrm{OPT}$. The operation in step (3) does not add to the weight of $S^\star$: when the boundary of a face F is added to $S^\star$, its weight is subtracted again when resetting the weights of the portal edges back to zero in step (7). □

If a brick is "ignored" by Thinning, the boundary of its enclosing mortar graph face is completely added to $S^\star$. Because $S^\star$ can be added to the final solution, every potential connection through that brick can be rerouted through $S^\star$ around the boundary of the brick. The interior of the brick is not needed.

An almost standard dynamic programming algorithm for bounded-treewidth graphs (cf. [AP89, KS90]) can be applied to G_{thin} and (T, χ). However, for the leaves of the tree decomposition that are added in step (6.2) of the Thinning procedure, the cost of a subset of portal edges is calculated as the cost of the minimum Steiner tree interconnecting these portals in the corresponding brick using [EMAFV87]. Since all the portal edges of this brick are present in this bag (recall that we do not delete parallel portal

edges after contractions), all possible solutions restricted to the corresponding brick will be considered. Since the contracted brick vertices only appear in leaves of the dynamic programming tree, the rest of the dynamic programming algorithm can be carried out as in the standard case.

Analysis of the running time. As was shown in [BKM09], the total time spent in the leaves of the dynamic programming is $\mathcal{O}(4^\theta n)$. The rest of the dynamic programming takes time $\mathcal{O}(2^{\mathcal{O}(\xi)}n)$. The running time of the thinning algorithm is dominated by the Contraction Decomposition Theorem 2.7 which is $\mathcal{O}_g(n \log n)$ [CC07]. Hence, the total time is $\mathcal{O}(2^{\mathcal{O}(\xi)}n + 2^{\text{poly}(g)}n \log n)$ for the general case; in particular, this is singly exponential in ε^{-1} and g, as desired. This proves Theorem 2.1.

2.3 Conclusion and Outlook

We presented a framework to obtain PTASes on bounded-genus graphs for subset-connectivity problems, where we are given a graph and a set of terminals and require a certain connectivity among the terminals. Specifically, we obtained the first PTAS for Steiner Tree on bounded-genus graphs running in $\mathcal{O}(n \log n)$-time with a constant that is singly exponential in ε^{-1} and the genus of the graph. Our method is based on the framework of Borradaile et al. [BKM09] for planar graphs; in fact, we generalize their work in the sense that basically any problem that is shown to admit a PTAS on planar graphs using their framework easily generalizes to bounded-genus graphs using the methods presented in this chapter. In particular, this gives rise to PTASes in bounded-genus graphs for Subset Tsp [BDT09], $\{0, 1, 2\}$-edge-connected Survivable Network [BK08b], and also Steiner Forest [BHM10].

As mentioned in Chapter 1 and Appendix A, H-minor-free graphs have earned much attention in recent years. Many hard optimization problems have been shown to admit PTASes and fixed-parameter algorithms on these classes of graphs; see, e.g., [DH05a, Gro03]. But subset-connectivity problems, specifically Subset Tsp and Steiner Tree, remain important open problems [Gro03, DHM07]. Both a spanner theorem and a contraction decomposition theorem are still missing for the H-minor-free case. Very often, results on H-minor-free graphs are first shown for planar graphs, then extended to bounded-genus graphs, and finally obtained for H-minor-free graphs. This is due to the powerful decomposition theorem of Robertson and Seymour [RS03] that essentially says that every H-minor-free graph can be decomposed into a number of parts that are "almost embeddable" in a bounded-genus surface. We conjecture that our framework extends to H-minor-free graphs via this decomposition theorem. The advantage of our methodology is that handling weighted graphs and subset-type problems are naturally incorporated, and thus it might be possible to combine all the steps for a potential PTAS into a single framework for H-minor-free graphs based on what we presented in this chapter. Hence, whereas our work is an important step towards this generalization, still a number of hard challenges remain; see also [DHM07] for a further discussion on this matter.

3 Dealing with Large Hidden Constants: Engineering a Planar Steiner Tree PTAS[1]

In the past few decades, a huge body of work has evolved that shows the existence of polynomial-time approximation schemes for many hard optimization problems on various input classes. These algorithms are, of course, of high theoretical importance, and many of them are seminal results. But unfortunately, most of these results are far from being applicable in practice; the problem is that, even though their theoretical running time is polynomial for a fixed error bound, often even near linear, the constants hidden in the big-$\mathcal{O}$ notation turn out to be much too large for an actual implementation, even for large approximation factors. We present *the first attempt on implementing such a highly theoretical algorithm, namely, a* PTAS *for the Steiner tree problem in planar graphs*, showing that it is possible to actually implement and run it, even on large instances, already today – but under some compromises. This suggests that some improvements, both in theory and practice, might make these great theoretical works bear practical fruits in the future. On a higher level, we would like to stimulate the theoretical world to further improve on the practical applicability of their results by showing that implementation attempts are not necessarily as far fetched as is generally thought today.

The compromises we had to make are of the following nature: even though we have implemented the algorithm completely and correctly and substantially accelerated it in several places, with today's processing power and space limitations, we are not able to set all parameters as high as required by the original algorithm; hence, we cannot maintain an a-priori guarantee on the approximation ratio. But our experiments show that, with our choice of parameters, we do get the desired approximation ratios suggesting that a much tighter analysis might be possible. Also, there is a natural trade-off between running time and solution quality.

On PTASes and Parameterized Complexity

Ever since Lipton and Tarjan's planar separator theorem [LT79], a large number of PTASes on various problems have been presented, see e.g. [Bak94, GKP95, AGK+98, RS98, Aro03, Gro03, Kle08, DH05a, Kle06]. Even though all these algorithms are polynomial for a fixed error bound, there is a qualitative difference in their running times that becomes very important when it comes to their practical applicability: whereas many PTASes are in XP when parameterized by $1/\varepsilon$, there have been great efforts in recent years in obtaining *efficient* PTAS*es* (EPTASes), namely, PTASes that are in FPT

[1] This chapter is based on joint work with Matthias Müller-Hannemann [TM09a].

with respect to the parameter $1/\varepsilon$, see e.g. [Bak94, RS98, Gro03, Kle08]. *Note that only* EPTAS*es come into question when it comes to implementation attempts.* In this sense, these efforts – and more generally, the design of FPT-algorithms – are extremely valuable and important in pushing such algorithms towards practicality. Of course, even after an EPTAS or an FPT-algorithm is discovered for a problem, it still remains to find the algorithm with the smallest dependence on the parameter; indeed, in our case, it was such an improvement on the parameter-dependence that made our implementation possible (see the next subsection). In a more general context, the current work can be seen as an implementation and engineering of an FPT-algorithm where the theoretical dependence on the parameter is huge – but, with some compromises, somehow manageable.

The Steiner Tree Problem

We consider the Steiner tree problem, which we thoroughly discussed in Chapter 1. Recall that it is NP-hard [Kar72] and even APX-hard [BP89] in general graphs but admits an FPT-algorithm when parameterized by the number of terminals [DW72, EMAFV87, BHKK07]. A 2-approximation can be easily found [Cho78, Ple81] in time $\mathcal{O}(m+n\log n)$ in general graphs [Meh88] and in linear time on H-minor-free graphs [TM09b]. The Euclidean and rectilinear Steiner tree problem are also NP-hard [GJ77, GGJ77] but admit a PTAS [Aro98, Mit99, RS98]. The rectilinear case can be reduced to the planar graph version by using the so-called *Hanan-grid* [Han66, GC94], and the Euclidean case can be reduced to planar graphs as shown in [MT10].

In planar graphs, the Steiner tree problem is also NP-hard [GJ77]. Erickson et al. [EMAFV87], Bern [Ber90], and Bern and Bienstock [BB91] showed that the problem is in FPT when parameterized by the genus and the number of faces or layers in which terminals appear. Very recently a PTAS has been found by Borradaile et al. [BKM09], which we reviewed in Chapter 1. While the first version of the PTAS [BKM07a] was *triply exponential* in a polynomial in the inverse of the desired accuracy, i.e. had a running time of $\mathcal{O}(2^{2^{2^{\text{poly}(1/\varepsilon)}}} \cdot n\log n)$, the complexity has been improved to a *singly exponential* algorithm running in time $\mathcal{O}(2^{\text{poly}(1/\varepsilon)} \cdot n\log n)$ in [BKM07b, BKM09]. This was an important step in making an implementation attempt possible. Still the exponent is a polynomial of *ninth degree*, which renders a direct implementation hopeless. Their result goes in line with several other highly theoretical papers that showed the existence of PTASes for important hard optimization problems, see e.g. [Bak94, Aro03, Gro03, Kle08, DH05a, Kle06], to name a few. To the best of our knowledge, no attempts on actually implementing these algorithms have been made to date.

Steiner tree is also a very important problem in practice and has to be solved in many industry applications, most prominently, in VLSI design. Hence, numerous implementations exist that are able to solve it, often very well, in practice. The most important exact algorithms are due to Zachariasen and Winter [ZW99] for geometric instances, Koch and Martin [KM98] using integer linear programming techniques, and Polzin and Daneshmand [PD01, PD02, PD06] with the strongest results for general graphs. Also, many powerful heuristics exist, see, for example, [KR92, GRSZ94, PW02, KMZ03].

The Mortar Graph and Its Uses

Recall that the concept of a *mortar graph*, as introduced by Borradaile et al. [BKM09] and defined in Section 1.5.1, is a grid-like subgraph of the input that has several useful properties, the most important of which are that it contains all the terminals, has bounded weight, and that there exists a near optimal solution that crosses each of its faces at most at a bounded number of vertices. The parts of the original graph that are enclosed in the faces of the mortar graph are called *bricks*. This mortar graph/brick-decomposition, in a sense, replaces the need for a spanner and is the centerpiece of the improved PTAS presented in [BKM07b, BKM09]. Very recently, it has been shown that it can also be used to approximate other problems: a PTAS for the minimum 2-edge-connectivity survivable network problem was given in [BK08b]; A PTAS for the Steiner forest problem has been announced in [BHM10]; and the traveling salesman problem is shown to admit this methodology in [BDT09]. As we saw in Chapter 2, the latter work actually generalizes the concept of a mortar graph to graphs of bounded genus and gives an outlook on even further generalizations. In the current chapter, we present the first implementation of a mortar graph/brick-decomposition for planar graphs and use it to obtain a PTAS for the Steiner tree problem. We hope that this implementation will prove useful for other problem domains, such as the ones mentioned above, as well.

Relation to Bounded-Treewidth Algorithms

The main idea of the PTAS of Borradaile et al. is to first find the mortar graph, then decompose it into parts of *bounded dual radius*, called *parcels*, and then apply *dynamic programming* to each of the parcels. The decomposition into parcels is in the nature of Baker's work [Bak94] and the dynamic programming is very similar to algorithms on graphs of bounded treewidth or branchwidth, see e.g. [BK08a]. In fact, planar graphs of bounded dual radius have bounded treewidth (though the situation here is somewhat different since we also have to deal with the bricks). Implementations of algorithms on graphs of bounded treewidth have been studied several times in the literature, see e.g. [ADN05] and [KvHK02]. Recent surveys are given by Hicks et al. [HKK05] and Bodlaender and Koster [BK08a]. These algorithms depend exponentially on the treewidth of the underlying tree decomposition and hence can only be applied if a tree decomposition of small width is known for the input graph. However, in our case, we only need to apply these algorithms to the parcels, which are parts of the mortar graph, which in turn can be (and usually are) much smaller than the input graph. Hence, we are able to attack very large problem instances, with up to one million vertices, see Section 3.3.

Contribution and Outline of this Chapter

In this work, we make a first attempt to bridge the gap between the theoretical world of approximation schemes and practice. Our aim is not to beat the current heuristics and exact solvers for the Steiner tree problem but *to present a new approach based on deep theoretical results, discuss its current limitations, and give an outlook for its possible future use*. We had to apply several modifications and non-trivial implementation

techniques to make an implementation possible at all. These techniques comprise a main part of our contribution and are described in Section 3.2; they build the main focus of this paper. Before we get to our implementation, we shortly describe the original PTAS and its main implementation challenges in Section 3.1. In Section 3.3, we present our experimental results, which well exceed our own expectations! In our experiments on FST-preprocessed instances[2] from the SteinLib library [KMV01], we are on average only about 1% away from optimum and on our randomly generated test instances, we are able to handle instances with up to one million vertices. To see how our implementation of the PTAS compares to one of the best heuristics, we also implemented a version of the batched 1-Steiner heuristic [GRSZ94]. While this heuristic still beats our method on many instances (within the range of parameters that we chose for our tests), we observe that on large instances, we are able to produce solutions that are nearly as good in much less time. Also, we obtain additional slight improvements by combining these two methods. We present a thorough experimental evaluation of the various parameters of the PTAS in order to gain a better understanding and a deeper insight into their empirical effects. Finally, in the last section, we give a short summary and an outlook on possible future work.

3.1 The PTAS and Its Challenges

In this section, we first review some properties of the PTAS of Borradaile et al. [BKM09] and its most important implementation challenges before we go on to explain our modifications and implementation techniques.

3.1.1 The Parameters of the PTAS

A detailed review of the PTAS along with formal definitions of the mortar graph and the bricks is given in Chapter 1. We briefly recall the most important concepts and parameters. Let the input graph be $G = (V, E)$, having n vertices and m edges, let $R \subseteq V$ be the given set of terminals, and OPT be the length of an optimal Steiner tree of R in G. We denote the mortar graph by MG and recall its most important properties below:

(i) $\ell(\text{MG}) \leq \alpha \,\text{OPT}$;
(ii) there exists a Steiner tree T for R in G such that $\ell(T) \leq (1 + c\varepsilon) \cdot \text{OPT}$; and
(iii) for each brick, the part of T in the interior of B intersects MG in at most θ vertices called *portals*.

Here, α and θ are constants that depend polynomially on ε^{-1} and c is an absolute constant that is not specified exactly. Mote that the properties mentioned above partially follow from the powerful Structure Theorem 1.7 by Borradaile et al. [BKM09]. Another parameter that plays into the mortar graph construction is κ, the *spacing of*

[2] These are rectilinear instances that were preprocessed with GeoSteiner [WWZ03], a software for computing full Steiner tree (FST) sets. For a description of the algorithm see [War97, WWZ00].

.	parameter usage (keywords)	theor. value	our strat.		typical values
α	weight-factor of MG	$9\varepsilon^{-1}$	actual val.	✓	1–3
κ	spacing of super-columns	$\mathcal{O}(\varepsilon^{-3})$	Sect. 3.2.1	✓	5–20
θ	# of portals	$o(\varepsilon^{-7.5})$	parameter	×	5–10
η	dual radius of parcels	$\mathcal{O}(\varepsilon^{-2})$	Sect. 3.2.2	✓	2–6
γ	error-factor of parcel decomp.	$\eta\alpha^{-1}\varepsilon$	parameter	✓	2–3
ξ	cut size/treewidth in DP	$2\theta\eta+1$	actual val.	✓	15-30
λ	max. table size of DP	$\mathcal{O}(4^{\xi})$	parameter	×	$10000/\varepsilon$
ε	total error	parameter	-	-	0.01–0.06

Table 3.1: A table of the parameters of the PTAS. The second column specifies the part of the algorithm in which the parameter is used; the fifth column specifies if our strategy could theoretically violate the approximation guarantee: a ×-sign means that this is the case; and in the last column we give estimates on values that we typically observed or used in our experiments.

the super-columns: recall that in order to construct a mortar graph, we first find a strip-decomposition, identify a number of columns in each strip and select every κth column to be a super-columns; the mortar graph consists of the strip-boundaries together with the super-columns. See Sections 1.5.1 and 3.2.1 for further details. The mortar graph and the bricks are illustrated in Figures 1.3 (a)–(c) on page 14. See also Figure 3.10 on page 65 for a large example of an actual mortar graph.

After MG is constructed, it is decomposed in the nature of Baker's approach [Bak94] into a number of parts, called *parcels*, with the property that each parcel has *bounded dual radius*, namely, at most equal to a parameter η. The total weight of the parcel boundaries is bounded by $\frac{\alpha}{\eta}\,\mathrm{OPT} \leq \frac{\varepsilon}{\gamma}\,\mathrm{OPT}$ for a constant γ if we choose $\eta = \alpha\gamma\varepsilon^{-1} = \mathcal{O}(\varepsilon^{-2})$. This way, one may add the parcel boundaries to the final solution without violating the approximation guarantee.

Finally, A dynamic programming (DP) algorithm is presented that finds optimal Steiner trees in the *brick-contracted graph* $\mathcal{B}^{\div}(P)$ for each parcel P (see Figure 1.3 (d)). It is shown that the size of the cuts that are considered in the DP – that we imprecisely but for convenience call *treewidth* of the instance – is bounded by a constant $\xi = 2\theta\eta + 1$. The DP runs in time exponential in ξ. Afterwards, the solutions of the parcels are connected using the parcel boundaries; and the Structure Theorem 1.7 by Borradaile et al. [BKM09] guarantees that the constructed solution is close to optimum. See Section 1.5 for further theoretical details and Section 3.2 for implementation-level details of the algorithm.

3.1.2 Main Challenges

The "official" running time of the PTAS is $\mathcal{O}(n \log n)$, not specifying the hidden constants. But in order to ensure that the solution is within a factor of $(1+\varepsilon)$ of the optimal

solution, the constants in the algorithm have to be chosen appropriately. Table 3.1 gives an overview of the involved parameters and constants of the algorithm and how they are/should be chosen.

As one can see, the constants tend to get extremely large for even fairly large values of ε, such as 0.5; and some of them are not even specified concisely. The most dramatic problem occurs in the dynamic programming: the table size is the number of non-crossing partitions of at most $\xi = 2\theta\eta + 1$ elements, which is by a Catalan bound within $\mathcal{O}(4^{\xi}) = \mathcal{O}(2^{\varepsilon^{-9.5}})$ [BKM09]. Now noticing that, say, the 15th Catalan number is already around 10 million, the 20th around 6.5 billion, and that in order to calculate the table entries for a node, each pair of table entries of the child nodes have to be considered, it is pretty much impossible to implement the PTAS as it is specified in theory, even if these constants are chosen to be extremely small. So, besides the challenge of constructing the mortar graph and the decompositions efficiently, the most important challenge of an implementation is to modify the algorithm and find compromises so as to *make the dynamic programming work in practice.*

3.1.3 Our Approach

An important observation that we made is that the constants specified in [BKM09] are *worst-case constants*; for a given instance in practice, one can compute a *lower bound* on the solution value and choose the constants according to this value. It turns out that, on all tested instances, the constants may be chosen to be much smaller. The only constant that cannot be chosen easily according to this rule is θ, the number of portals on the brick boundaries. In fact, in order to guarantee a $(1+\varepsilon)$-approximate solution – according to the analysis given in [BKM09] – this constant has to be chosen to be very large. We decided to choose this value empirically; it turns out that even for very small values of θ, like 5 or 8, the solution is well within the required approximation bounds – in fact, oftentimes very close to optimum (see Section 3.3)! The dynamic programming tables still become very large. In order to get reasonable running times, we decided to introduce a new parameter λ to set a bound on the maximum allowed size of the DP-tables. We choose this parameter as a function of ε to reflect the need for larger table sizes as ε decreases. In order to achieve good solutions, even with these bounds, we employed several techniques that are described in the following subsections. The running time of our implementation is $\mathcal{O}(n^2 \log n + n\lambda^2)$ where the first term originates from the mortar graph construction and the second term from the dynamic programming. But as can be seen in Section 3.3, the empirical running time of our DP is well below what one would expect from $\mathcal{O}(n\lambda^2)$.

3.2 Our Implementation

We break down the description of our implementation into four main parts: the construction of the mortar graph, the decomposition into parcels, the dynamic programming, and the lifting, i.e. putting everything together and creating a valid solution. In what follows, each of these stages is discussed in detail.

3.2.1 Constructing the Mortar Graph

We refer to Section 1.5.1 for a detailed algorithmic description of the mortar graph construction along with figures. In what follows, we complement this description by giving some implementation-level details and considerations.

Main Data Structures We implemented the program in C++ using the Standard Template Library (`STL`). We defined a structure called `EmbeddedGraph` as our main data structure that stores a combinatorial embedding of a graph; it contains three main arrays, one for the edges, vertices, and faces of the graph, respectively. The input is undirected but each edge is stored as a pair of directed edges having a pointer to each other. Each directed edge knows its head and tail vertices and the faces to its left and right. A directed edge (v_0, w_0) has a pointer to its next and previous vertex edges, (v_0, w_1) and (v_0, w_{-1}), regarded in clockwise order, and also a pointer to its next and previous face edges, (w_0, v_1) and (v_{-1}, v_0), counted in counter-clockwise order, where an edge is always associated with the face on its left. Each vertex has only a pointer to its first outgoing edge and its degree. Likewise, each face has only a pointer to its first edge and its degree, i.e. the number of edges on its (not-necessarily simple) facial walk. The input may be given "face-wise", i.e. specifying the facial walks in counter-clockwise order, or "vertex-wise", i.e. specifying the adjacent edges of each vertex in clockwise order. Note that with this data structure, we have access to the primal and dual of the graph simultaneously since we have defined them on the same set of edges: a directed edge that connects two vertices in the primal is the same edge that connects the face to its left with the face to its right in the dual. We also make use of an additional data structure named `FastGraph` that simply stores a graph as an adjacency list. In the following, we denote the weight of an optimal solution by OPT and the weight of a lower-bound, by LB. We compute LB as half the weight of a 2-approximation.

Creating a Split 2-Approximation The first step of the mortar graph construction is to find a 2-approximate Steiner tree of the input. We accomplish this task using Mehlhorn's algorithm [Meh88] in $\mathcal{O}(n \log n)$ time (see Section 1.2). The 2-approximate Steiner tree has to be split along its Eulerian cycle, in which each edge is traversed exactly once in each direction, thus adding a new face to the graph, which may be regarded as the *outer-face* of the graph (see Figure 1.4 (a)–(c)). We create a copy of the input and split the tree in this copy in linear time (having a copy not only simplifies restoring the original solution but is also important in the next step).

Decomposition into Strips We start with the whole graph and the boundary of its outer-face, as described above; if there exists a subpath of the current outer boundary that is not ε-short, we determine a smallest subpath violating the condition, find a shortest path between its endpoints, and separate it from the current graph, storing it as a new strip (see Figure 1.4 (d)). Klein [Kle06] describes an $\mathcal{O}(n \log n)$-time algorithm for the strip decomposition using dynamic trees (see, e.g. [TW05]) and his multiple-source shortest-paths algorithm [Kle05]. We decided to follow the main parts of the

algorithm as given in [Kle06] but instead of using dynamic trees, which cause a large constant overhead, we used Dijkstra's algorithm [Dij59] once from each relevant vertex, resulting in a quadratic-time algorithm. This implementation is fast enough for (almost) all of our instances. An important implementation detail is that the shortest paths that are used to separate the strips might actually overlap with some parts of the current outer boundary and thus create tree-like parts on the boundary; these parts have to be handled appropriately in an implementation. Note that since this algorithm separates the strips from the graph, it gradually destroys it. We store the strip boundaries in a table and throw away the split copy of the graph after this step is finished.

Adding Super-Columns The next step is to divide each strip into a number of bricks by adding super-columns. In each strip, we find a shortest path from every vertex on the south boundary to (any vertex on) the north boundary. This can be done by a single shortest-paths computation in each strip in $\mathcal{O}(n \log n)$ total time [Kle06]. From this, we extract the set of columns and super-columns according to a parameter κ as described in Section 1.5.1 (see Figures 1.4 (e)–(d)). This way, it is ensured that each brick contains at most κ columns, and furthermore, that the total weight of the super-columns is at most κ^{-1} times the total sum of all columns.

The effect of the parameter κ is two-fold: on one hand, the larger κ is, the larger will be the bricks and so, with a fixed number of portals, the best possible solution in the portal-connected graph might become worse; on the other hand, the smaller κ is, the larger is the sum of the weights of the super-columns, which are added to the final solution in the theoretical analysis. In [BKM09], κ is chosen as $\mathcal{O}(\varepsilon^{-3})$ so as to guarantee that the sum of the weights of the super-columns is at most $\mathcal{O}(\varepsilon \operatorname{OPT})$. We choose κ as follows: let S_C be the total sum of the weights of all columns; we set $\kappa_0 = \frac{3S_C}{\varepsilon \operatorname{LB}}$; in each strip, we determine the smallest $\kappa \leq \kappa_0$ such that the sum of the weights of the resulting super-columns is at most κ_0^{-1} times the sum of the weights of the columns in that strip. This way, we guarantee that the sum of the weights of all super-columns is at most $\kappa_0^{-1} S_C \leq \frac{\varepsilon}{3} \operatorname{OPT}$ while choosing the κ for each strip as small as possible. Additionally, we benefit from the fact that we do not automatically add the super-columns to the final solution and so do not necessarily have this extra-weight added to our solution.

Constructing the Mortar Graph and the Bricks We keep a boolean vector that specifies for each edge of G if it is included in the mortar graph or not (this vector is filled in the steps above). Then we construct the mortar graph as a new `EmbeddedGraph` "vertex-wise" by going over the vertices of G and adding their adjacent mortar edges in clockwise order while keeping maps between original and mortar vertices and edges. Afterwards, we scan each face of the mortar graph and determine if the corresponding part in G includes some edge that is not a mortar edge; in this case, we have found a new brick graph, which we store as a `FastGraph` since its embedding is not needed later on. The theoretical algorithm requires that the boundaries of the bricks are split so that each one forms a simple cycle that corresponds one-to-one with its mortar face boundary; this way, it is ensured that the portals lie on the outer boundary of the brick. We

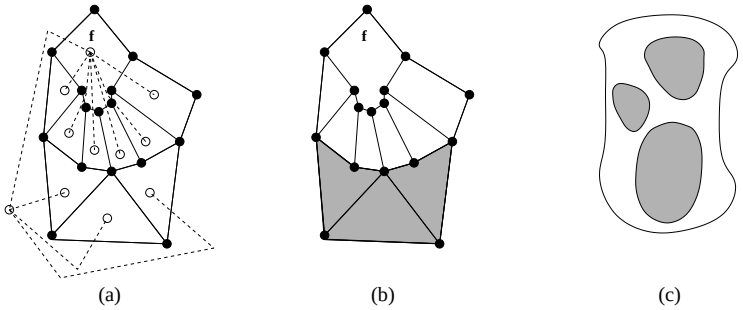


Figure 3.1: (a) A planar graph with black vertices and solid lines and a breadth-first-search tree of the dual represented with white vertices and dashed lines; (b) a parcel decomposition corresponding to this BFS tree, rooted at face f, separating after level 1 of the tree; the white faces constitute the first parcel and the gray faces the second; (c) an illustration of a more complicated situation where many parcels are created at only one separating level.

found out that this step is not necessary in practice and that it bears no disadvantage to store the bricks without splitting their boundary and allowing some portals to be placed inside the brick (which can only happen if the corresponding mortar face has a tree-like part that goes inside the brick, cf. Section 3.2.3). See Figure 3.10 on page 65 for a large example of a mortar graph.

Designating Portals The original portal selection algorithm is straightforward: for a brick B, calculate the weight of the mortar boundary ∂B; start at some vertex on the boundary and choose it as the first portal v_0; then, walking along the mortar boundary, whenever the weight of the current path exceeds $\ell(\partial B)/\theta$, choose the current vertex as the next portal and reset the current path to zero. Our implementation differs in two aspects: first, we only select such vertices as portals that are adjacent to at least one non-boundary edge inside the brick; second, if we are to select a vertex that is already chosen as a portal for this brick, we skip it and reset the current path length to zero — there is no benefit in selecting a vertex as a portal twice. In the original algorithm, θ has to be $10\varepsilon^{-2}o(\varepsilon^{-5.5})$ but as we discussed above, we choose the value of θ empirically, usually between 5 and 10.

Adding Portal Edges In the original algorithm, it is required to augment the mortar graph as follows: for each face F of MG that corresponds to a brick, add a *brick vertex* to F and connect it to the portals of F via zero-weight edges (this corresponds to the operation $\mathcal{B}^{\div}$ in the theoretical description of the algorithm). We do not perform this step explicitly; we add these *portal edges* to the graph, but do not incorporate them into our `EmbeddedGraph` data structure. Instead, we store them in extra tables and treat them as special edges. These edges play an important role in the dynamic programming part of the algorithm.

3.2.2 Decomposition into Parcels

The parcel-decomposition algorithm is in the nature of Baker's decomposition [Bak94] applied to the mortar graph: perform breadth-first search (BFS) in the dual and label the BFS-layers periodically by $0, \ldots, \eta - 1$; this partitions the edges into η sets, and we choose the one with the smallest total weight to be the set of parcel boundaries. Each part of MG that lies between two consecutive parcel boundary levels is called a parcel and is defined to include the boundaries (see Figure 3.1 (a)–(b)). Note that there might be several parcels lying between the same two consecutive levels (see Figure 3.1 (c)). The radius of the dual of each parcel is by construction bounded by η. Borradaile et al. [BKM09] define η to be $18\varepsilon^{-2}$. In practice, we cannot afford η to be more than about 5. The trick to achieve this is as follows: whereas in theory, it does not matter at which vertex to root the tree, in practice, we search for a root-face that minimizes the depth of the resulting BFS tree; furthermore, we are allowed to count faces that only share a vertex together also as adjacent (cf. [Tam03]), see the next paragraph and Figure 3.2. These two techniques alone often result in a very small tree depth so that no parcel-decomposition is necessary at all. Additionally, we apply the following: instead of first setting the value of η, we take the error factor γ as a parameter and calculate the maximum allowed parcel boundary size PB as $\frac{\varepsilon \, \mathrm{LB}}{\gamma}$. Then, starting at the mid-level of the BFS tree, we look in both directions for one level at a time and select the first level whose total edge-weight is not more than PB. This way, we try to split the tree as evenly as possible at some level that complies with the given weight-bound, if such a level exists. Currently, we have not implemented a strategy that tries to choose more than one separating level for the parcels since our strategy is sufficient for our test cases, and we do not expect it to be often possible to choose more than one separating level while remaining within the given bound PB. We store each parcel by storing the status of each edge of MG in the parcel, without making an explicit new copy of the graph.

About Vertex-Degrees The original algorithm assumes that each vertex of G has maximum degree 3. This can be achieved easily by splitting vertices of higher degree and creating zero-weight edges. We chose to not split the graph initially; we only split the vertices when it is needed. Firstly, this decision considerably accelerates the mortar graph construction; additionally, it turns out to be very useful in the parcel decomposition: as already mentioned above, when searching for a BFS-tree with minimum depth, we are allowed to *cross* vertices, counting faces that only meet at a vertex as adjacent (see Figure 3.2 (a)–(b)). This results in trees of considerably smaller depth. After the tree with minimum depth is selected, we traverse the tree and split the vertices at the points where they are crossed, creating an actual zero-weight edge that makes the corresponding faces adjacent (see Figure 3.2 (c)). So, after these splittings, our chosen tree becomes an actual BFS tree of the dual graph. Similarly, after selecting the parcel boundary levels, we split the vertices on these levels to ensure that the boundary of each parcel becomes a simple cycle. In other parts of the algorithm, we also only split vertices when it is really needed to have bounded degree and mention it wherever applicable.

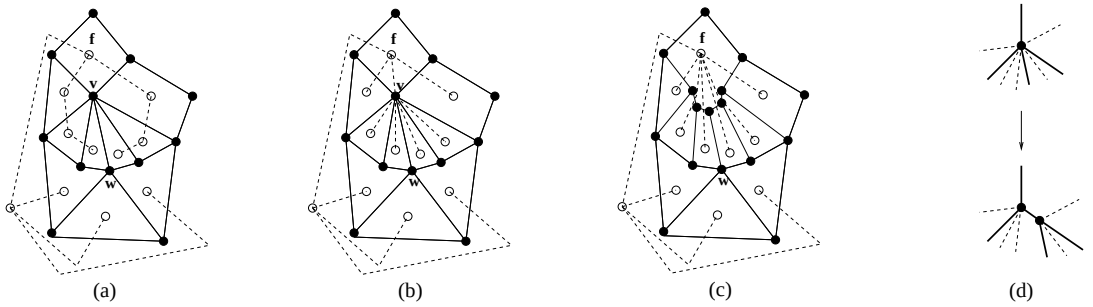


Figure 3.2: A planar graph with black vertices and solid lines and a breadth-first-search tree of the dual, represented with white vertices and dashed lines; (a) the depth of the dual BFS-tree rooted at face f is 3; (b) if "crossing vertices" is allowed, the depth of the tree becomes 2; (c) after splitting vertex v accordingly, the tree becomes a valid dual tree of depth 2; note that, e.g., vertex w needs not be split; (d) splitting a vertex in the primal tree: the solid edges are in the primal tree and the dashed edges are not; it is only necessary to establish a maximum degree of 3 w.r.t. to the primal tree edges.

Constructing the Primal Trees and Preparing the Parcels It is a well-known fact in graph theory that the set of edges not included in a spanning tree of the dual of a planar graph builds a spanning tree in the primal planar graph (see Figure 3.3 (a) and Observation 2.3). We consider this *primal tree* for each parcel P and add the first portal edge of each brick to it to obtain a primal tree for $\mathcal{B}^{\div}(P)$. We make sure that every vertex of this tree has degree at most 3 by splitting the vertices of higher degree using zero-weight edges if necessary (see Figure 3.2 (d)). If the parcel contains a terminal, we root the tree at a terminal. We add an auxiliary root edge to the tree to ensure that the final solution of the parcel is connected. In the original PTAS, it is stated that the weight of the parcel-boundaries has to be set to zero and an algorithm is given to add some terminals to these boundaries so as to ensure that, at the end, we get a connected solution by adding the parcel boundaries to the solution. We do set the boundary weights to zero but decided to not add new terminals; we describe in the last part of this section how we create a connected solution while trying to avoid including the whole parcel boundaries as originally specified.

3.2.3 Dynamic Programming

The dynamic programming part of the PTAS is based on the following observation: for a directed edge $e = (v, w)$ of the primal tree of a parcel, consider the subtree T_e of the primal tree rooted at w; this subtree is separated from the rest of the graph by a cut C_e calculated as follows (see Figure 3.3): consider the faces F_L and F_R to the left and right of e; the path connecting F_L and F_R in the dual BFS-tree is the desired cut and has at most 2η edges since the depth of the tree is η. But each face may contain up to θ portal edges, so the total size of C_e, including e itself, is at most $2\theta\eta + 1$, which is a

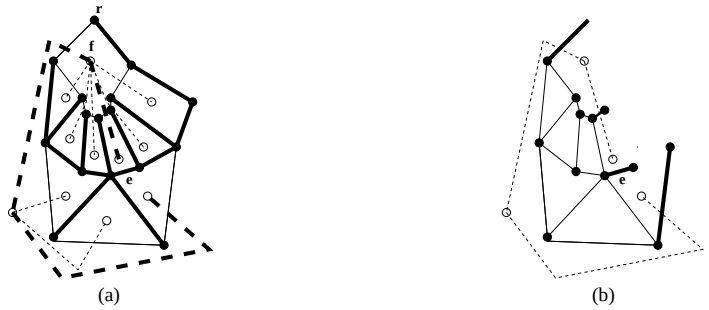


Figure 3.3: A planar graph with black vertices and solid lines and a breadth-first-search tree of the dual represented with white vertices and dashed lines; (a) the primal tree rooted at vertex r is indicated in bold solid lines; the path connecting the left face of edge e to its right face in the dual BFS tree is indicated in bold dashed lines; (b) the subproblem corresponding to edge e; the cut C_e that separates this subproblem from the rest of the graph is drawn with bold solid lines.

constant. Hence, in theory, we may enumerate over all possible non-crossing partitions of C_e and calculate the optimal solution of the partition via dynamic programming. The optimal solution of a partition is a forest of minimum weight such that each terminal of T_e is included in the forest and each tree of the forest has at least one edge in C_e. If T_e contains the first portal edge of a brick, that brick has to be considered in the solution for T_e, too. The value of a solution is twice the sum of the weights of edges of the solution inside T_e plus (once) the weight of the edges selected from C_e. At the end of the DP, the final solution value will be divided by 2. The details of the dynamic programming and its implementation are given below.

Dealing with Partitions In the current state of our program, we assume that the largest cut size that we have to deal with is 64 (which is the case in all of our test cases; in fact, sets of size more than 30 are already very hard to deal with since the number of their partitions is extremely large). We store a subset of edges as a bitset in an unsigned 64-bit long integer. A partition is stored as a tuple of such bitsets where the first set specifies which edges are included in total in this partition. We call this first set the *inclusion set* of the partition. Each subsequent set specifies one part of the partition. Besides the first set, the remaining sets are always sorted from large to small according to their unsigned integer value.

The Base Case The base case of the dynamic programming occurs at the leaves of the primal tree. If a leaf is an edge $e = (w, v)$ of MG, we simply enumerate over all subsets of non-zero edges adjacent to v and store the solution together with the zero-weight edges. If v is a terminal, we make sure that all stored solution sets are non-empty; otherwise, we make sure that an empty solution, without the zero-weight edges, is also stored.

If the edge e is a portal edge of a brick B, we proceed as follows: first, for all subsets of

portals of B, we calculate the optimal solution via a dynamic programming algorithm for Steiner tree in planar graphs with all terminals on one face by Erickson et al. [EMAFV87]. This algorithm runs in time $\mathcal{O}(\theta^3 \cdot n + \theta^2 \cdot n \log n)$; however, determining the optimal Steiner tree for *all* subsets of portals requires $\mathcal{O}(2^\theta \theta^2 \cdot n \log n)$ time. Our case is slightly different from Erickson et al. [EMAFV87] and from Borradaile et al. [BKM09] since we did not split the boundary of the bricks, and therefore not all portals lie on the outer face. But this is not a problem since the non-boundary portals can be mixed in via the general DP algorithm for Steiner trees of Dreyfus and Wagner [DW72] that runs in total time $\mathcal{O}(3^\theta \cdot n \log n)$ for *all* subsets. If θ_b is the number of boundary portals, θ_n is the number of non-boundary portals, and $\theta = \theta_n + \theta_b$, the total time our algorithm requires to find an optimal Steiner tree for *all* subsets of the portals is $\mathcal{O}(3^{\theta_n}\theta_n \cdot 2^{\theta_b}\theta_b^2 \cdot n + 2^\theta \cdot n \log n)$. This is between the two running times mentioned above but usually much closer to that of [EMAFV87] since the number of non-boundary portals is usually small. Also, recall that θ can practically not be chosen to be more than 10 and is usually set equal to 5 (cf. Section 3.3); hence, this algorithm is indeed quite fast in practice.

After the solution for each subset is calculated, we look at each non-crossing partition of the portals (which are precomputed and stored for each $i = 0, \ldots, \theta$) and store as its value the sum of the values of the parts it contains. An important improvement in this stage is that we only store the solutions of such partitions of the portals for which the optimal Steiner trees of the parts are disjoint; if the trees intersect, they actually impose another partition on the portals that might even have a better solution and is considered at some point of its own. Even though checking the disjointness of the trees causes some computational overhead, this measure usually reduces the number of stored solutions considerably and results in a significant reduction of the total running time.

Solving the DP At a non-leaf edge $e = (w, v)$, the subproblem of T_e contains one or two subproblems T_{e_1} and T_{e_2} corresponding to edges $e_1 = (v, v_1)$ and $e_2 = (v, v_2)$ of the primal tree. Let C_0 be the set of edges adjacent to v, C_{e_1} and C_{e_2} be the cuts corresponding to T_{e_1} and T_{e_2}, and C_e be the cut corresponding to T_e (see Figure 3.4 (a)). In order to construct the DP table for T_e, the original algorithm tells that one has to look at every triple of partitions (S_0, P_1, P_2) where S_0 is a subset of C_0, P_1 is an entry from the solution table of T_{e_1}, and P_2 is an entry from the solution table of T_{e_2}; if some triple (S_0, P_1, P_2) is *consistent*, as defined below, then one calculates the *merged partition* resulting from merging S_0, P_1, and P_2 and stores it along with its value in the solution table of T_e. For $i = 1, 2$, let S_i be the inclusion set for P_i; a triple is considered consistent if: (i) for each pair $(i, j) \in \{0, 1, 2\}^2$, we have that $S_i \cap (C_i \cap C_j) = S_j \cap (C_i \cap C_j)$, i.e. the partitions have the same set of edges interconnecting v and the subproblems; and (ii) each subset in the merged partition of S_0, P_1, and P_2 contains an edge from the outer cut C_e; see Figure 3.4 (b). The latter condition ensures that each subset reaches the outside and will eventually be connected to the solution tree.

For an efficient implementation, we proceed as follows: for each edge of the primal tree, we store a *partition table* and a *solution table* (as C++ `STL-vectors`). For the base cases of the DP, the values for these tables are computed as described in Section 3.2.3

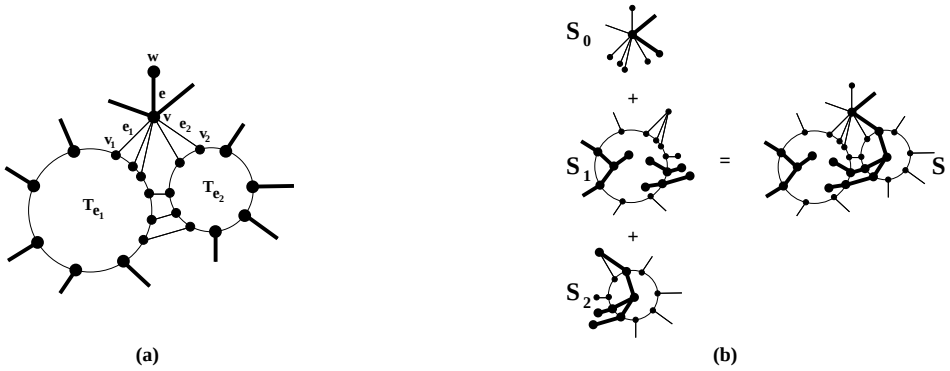


Figure 3.4: (a) Illustration of a nonleaf edge $e = (w, v)$ in the DP with children $e_1 = (v, v_1)$ and $e_2 = (v, v_2)$; the cut C_0 is the set of all edges adjacent to v, the cut C_e is indicated with bold lines, the cut C_{e_1} consists of all edges attached to the subproblem T_{e_1}, and the cut C_{e_2} consists of all edges attached to the subproblem T_{e_2}; (b) a triple of consistent solutions S_0, S_1, and S_2 indicated with bold lines and the result of merging them into a solution S for T_e.

and sorted in such a way that the entries of the tables correspond to each other. When solving an inner node T_e, we first collect the cuts C_e, C_0, C_{e_1}, and C_{e_2} and store them in an array called `CutMap` so that each edge is stored exactly once. We make sure that in the `CutMap`, the order of edges, from right to left, is as follows: (1) the outer edges C_e; (2) the remaining edges of C_{e_2}, i.e. $C_{e_2} \cap (C_{e_1} \cup C_0)$; (3) the remaining edges of C_{e_1}, i.e. $C_{e_1} \cap C_0$; (4) the remaining edges of C_0. Recall that our data structure for partitions is basically a collection of bitsets, where the first bitset is the inclusion set of the partition. Now, if we could make sure that the order of the edges in these bitsets corresponds to the current `CutMap` and order the partition tables according to their inclusion set, all partitions of C_{e_1} that include the same subset of C_0 would appear consecutively because the corresponding bits are the leftmost bits of the inclusion sets; similarly, all partitions of C_{e_2} that include the same subset of $C_0 \cup C_{e_1}$ would appear consecutively; in particular, we could use *binary search* to look for consistent triples! All we have to do is to remap the partition table of the left and right child to correspond to the current `CutMap` and sort them (along with their solution tables) according to the unsigned integer value of their inclusion sets.

Now, in order to find consistent triples, we can first go over all possible choices of S_0 as described in the base case of the DP and build the corresponding bitmask. Then we can find the first position in the table of T_{e_1} that matches S_0 by binary search and examine consecutive entries of the table until the first mismatch. Each examined entry corresponds to a partition P_1 that is consistent with S_0; we build the merged bitmask of S_0 and S_1 and look it up by binary search in the table of T_{e_2}. This way, we can identify consistent triples very efficiently.

After a consistent triple has been found, their subsets have to be merged to form a partition for T_e. We implemented a simple bit-manipulating merging algorithm to

accomplish this. If the number of the subsets is k and the number of edges of the `CutMap` without C_e is b, our implementation runs in time $\mathcal{O}(\min(k, b)k)$. Since both k and b are constants, this is a constant running time (recall that we do not allow the `CutMap` to have size more than 64 and that its real size is usually much less).The value of the solution is the sum of the values of the solutions chosen in the triple. This way, each edge internal to T_e is counted twice and each edge in C_e is counted once, as desired.

Managing the Partition Table As already mentioned, we set the maximum allowed table size to be λ. At each node, we only store the λ solutions with the smallest value. In order to manage the table efficiently, we use a balanced binary search tree (a C++ `STL-map`) to store the set of partitions currently found for this node. This way, when a new partition is found, it is easy to check if it is already contained in our table, if its value needs to be updated, and if it may be inserted into the table at all due to the mentioned size constraints. We also keep a reverse map, indexed by the solution values, so that the value of the longest stored solution can be found quickly. After a node is completely processed, these maps are copied into arrays (i.e. the `STL-vector`s of the partition and solution tables mentioned earlier) and emptied to be reused for the next node. Also, the partition tables of the children of the current node may be freed at this point in order to save memory (but not the solution tables since they are needed for reconstructing the solution).

Ensuring Good Solutions Keeping only the λ smallest solutions might not always result in very good solutions. In fact, it might happen that at later stages of the DP, no solution can be found at all. In order to make sure that (good) solutions are found, we always include the partition that corresponds to the 2-approximation in our partition table. In order to do so efficiently, we first have to do some preprocessing: we consider the intersection of the 2-approximation with the current parcel and call this forest T_2; we eliminate zero-weight cycles in the parcel by deleting zero-weight edges that are not part of T_2 and then augment T_2 by all the zero-weight edges that are connected to it. Note that since the parcel boundaries have zero-weight, this step makes sure that T_2 becomes a tree. We label the nodes of T_2 in a post-order traversal and additionally, store at each edge, the smallest of all labels that occur in its subtree. Using these labels, we can easily decide in constant time whether two nodes remain in the same partition of T_2 if an edge of T_2 is removed. This information can be used, in turn, to calculate the partition corresponding to T_2 at each node of the dynamic program in constant time using bit manipulations (in fact, linear in the number of edges in the `CutMap`). This calculation needs to be done only once per node of the primal tree; afterwards, we just have to make sure never to throw away the smallest-value solution that corresponds to this partition from our table. Adding this feature to our program is one of the main reasons why it works so surprisingly well, see Section 3.3. The value of the 2-approximation intersected with the current parcel also serves as an upper bound: any solution with a larger value may be neglected.

Handling Portal Edges Portal edges pose a special complication to the dynamic programming: on one hand, they are zero-weight edges, so one would not want to waste time and space in enumerating all of their subsets at each node; on the other hand, if they are always included, they are only consistent with partitions that also include them and hence require solutions to go through bricks, which might (and usually does) violate optimality; or if handled somewhat differently, they might impose every portal to be connected to the outer cut, which also violates optimality. We have developed several strategies to deal with portal edges but currently, the basic strategy of enumerating over all subsets of portal edges, i.e. treating them as non-zero edges, is our fastest and best strategy.

3.2.4 Creating a Connected Solution

After having solved each parcel separately, we have to make sure that our solution is connected. At worst, we could include the complete parcel boundaries, as specified in [BKM09]. But we decided to take the disconnected solution as it is and connect it as follows: first, we reduce the number of connected components by (conceptually) contracting each connected component and starting to build the minimum spanning tree of the distance network of the contracted graph by a method very similar to Mehlhorn [Meh88] (actually, we just set the weight of edges included in the solution to zero, instead of contracting them). When the number of connected components is reduced to some predefined constant (currently 5), we find the optimal Steiner tree of the remaining components using the algorithms of [EMAFV87] and [DW72] as described in the base case of our DP for the bricks.

This way, our solution is always better than the originally proposed method of simply adding *all* parcel boundaries; and since we are usually very close to optimum, accepting this computational overhead does make a significant difference in the quality of our solutions.

3.3 Experimental Evaluation

To evaluate our implementation we used test instances from the instance library SteinLib [KMV01] as well as randomly generated test instances. From SteinLib, we focus on results from FST preprocessed rectilinear graphs (cf. Section 3); optimal solutions are known for these instances and can be used for comparison. The instances have $10, 20, \ldots, 100, 250, 500$, and 1000 terminals, and each size class contains 15 instances. Since the instances are FST preprocessed, the total number of *vertices* of the instances are only up to 3 times as many as the number of terminals (as opposed to a quadratic number obtained from the Hanan-grid) and are very similar for each fixed number of terminals; hence, we report on these instances by their number of terminals instead of number of vertices. We use these instances to thoroughly investigate the influence of the various parameters of the algorithm.

We also shortly report on the LIN class from SteinLib that consists of a number of VLSI-derived instances where the optimal solutions are known as well. In the end

of this section, we present our results on large randomly generated instances. All our computations were executed on an Intel(R) Core(Tm) 2 Duo processor with 3.0 GHz and 4 GB main memory running under Ubuntu 2.6.22. Our C++ code has been compiled with g++ 4.1.2 and compile option -O2.

We compare our method against the standard 2-approximation (which can always be computed in at most a few seconds) and the well-established batched 1-Steiner (1-St) heuristic, which is known to produce near-optimal solutions on many instance classes [GRSZ94]. In spite of many heuristics and a few exact codes for the Steiner tree problem, implementations are either not publicly available or are tailored to geometric versions of the problem but not applicable to planar graphs. The idea of the 1-Steiner heuristic is as follows: we start with a 2-approximation and for every vertex not in the tree, we calculate its *gain*, i.e. the improvement obtained if the vertex is added as a fixed Steiner point to the set of terminals; the vertex with the highest gain is added to the tree and the procedure is repeated. In the *batched* version, we simultaneously add all vertices with *independent* positive gains in each iteration; two vertices u, v have independent gains if the gain of u does not change after v is added to the tree. Our implementation of 1-Steiner runs in $\mathcal{O}(n^2 \log n)$-time per iteration on planar graphs. The number of iterations is at most 4 in practice.

Unless otherwise mentioned, we use the following standard parameter settings in our tests: we set $\varepsilon = 0.5$, the number of portals $\theta = 5$, the table size $\lambda = 10000/\varepsilon$, the super-column parameter κ as described in Section 3.2.1, and the parcel decomposition parameter $\gamma = 2$.

3.3.1 Varying ε

Figure 3.5 shows the average gap and time for the FST preprocessed rectilinear instances of SteinLib for $\varepsilon = 1, 0.5, 0.3, 0.1$, and 0.05, respectively. The average is taken over all 15 instances of SteinLib for each fixed number of terminals. We can see that the solution quality improves monotonically with ε and that there is a nice trade-off between computation time and solution quality. Note that the optimality gap is below 2.2% in all cases, whereas the original "approximation guarantee" would be a gap between 5%–100%. Our algorithm well outperforms the standard 2-approximate solution: whereas the 2-approximation has optimality gaps above 5% on all cases (not drawn in the figure), our solution is well below a gap of 1% for $\varepsilon \leq 0.1$. One can also see that the 1-Steiner heuristic mostly lies between the solutions for $\varepsilon = 0.1$ and $\varepsilon = 0.3$ but outperforms our algorithm on the instances with 1000 terminals. On these instances, the heuristic takes at most 9 seconds and is thus clearly the better choice. We also combined our method with 1-Steiner by taking its solution as the initial tree for our mortar graph construction. This way, we can improve upon the 1-Steiner solution by up to 2.4% in some cases, but only slightly on average.

The slight irregularities for the cases with less than 100 terminals result from the fact that the solution quality and time depend on the structure of the mortar graph and its treewidth (cf. Section 3.3.4). It might be that some instance with fewer terminals has a more complicated mortar graph with higher treewidth and hence pushes the average

optimality gap and time of a particular class higher than a class with more terminals.

3.3.2 Increasing the Number of Portals

The parameter θ that specifies the number of portals in each brick is perhaps the most delicate parameter of this PTAS; it is the main reason why we have to give up the a-priori guarantee of the algorithm in our implementation. In theory, it has to be huge, namely about $\mathcal{O}(\varepsilon^{-7.5})$; in practice, even setting it to zero would be basically fine since already the 2-approximation is usually within a few percent of the optimum. On the other hand, the table size of the dynamic programming part of the algorithm grows exponentially with θ, and since we set the table size limits manually, we would have to choose this limit to be very large – even for fairly small values of θ – in order to actually benefit from increasing this parameter; but this, of course, would make the running time and memory requirement explode. Judging from our empirical observations (cf. Section 3.3.1 and Section 3.3.8), it seems that our compromise solution of setting $\theta = 5$ results in very good solutions and reasonable running times. The experiments in this section support this hypothesis.

We tested the FST instances from SteinLib with up to 250 terminals for $\varepsilon = 0.5$ and $\varepsilon = 0.1$ with $\theta = 5$ and $\theta = 8$. In order to minimize the effects of varying (and too small) table sizes, we set the maximum table size for all these tests equal to one million[3]. The results and a comparison to the 1-Steiner heuristic are given in Figure 3.6 (a) and Table 3.3.

As one can see, the average optimality gaps are all well below 1%, even for $\varepsilon = 0.5$ with $\theta = 5$. Decreasing the value of ε has a much stronger effect on the solution quality of these instances than increasing θ: the results for $\varepsilon = 0.1$ are (almost) consistently much better than for $\varepsilon = 0.5$. But for each fixed ε, the solutions do get better for $\theta = 8$ (this is not completely trivial since the table size limit could still distort the results). The batched 1-Steiner heuristic needs at most 2 seconds on these instances and performs somewhat better than the runs with $\varepsilon = 0.5$ but is outperformed when setting $\varepsilon = 0.1$. The increase of the running times is not completely consistent in all cases since it depends too much on the achieved structure and treewidth of the constructed mortar graphs. But in most cases, especially the larger instances, the runs with $\theta = 8$ take considerably longer than the ones with $\theta = 5$ – regardless of the value of ε. This is indeed expected since the treewidth should grow linearly with θ; and even the slightest increase in the treewidth can cause much longer running times (cf. Section 3.3.4).

We would also like to mention that within each FST preprocessed class of SteinLib with a fixed number of terminals (remember, we take averages over 15 instances in each class), there are some considerable discrepancies between the solutions of the various instances, especially the running times. This is, again, due to the fact that different instances (and different parameter settings) can have very different mortar graphs and treewidths and thus require very different running times to be solved. In order to give a more detailed account on this observation, we present the individual results for the

[3]We also ran some tests with our standard settings for the table sizes but the results were highly distorted; further experiments regarding the table sizes are given in the next subsection.

instances with 250 terminals in Table 3.4. As in all of this subsection, the table size used in these tests are constantly set to one million. Luckily, as we saw in the experiments so far, when we build averages on these classes, we do observe (mostly) consistent behavior.

3.3.3 A Closer Look at the Table Size

The actual required table size for the dynamic programming is proportional to the Catalan number of order of the treewidth [BKM09]. Of course, we cannot afford such huge dynamic programming tables; so, we had to introduce the parameter λ to specify the maximum allowed table size. In our standard parameter setting, we set it equal to $10000/\varepsilon$ to reflect the need for larger table sizes with decreasing ε; naturally, this is not really sufficient but it does work reasonably well in practice, cf. Section 3.3.1. But it still causes unpleasant distortions like worse solutions for smaller values of ε or larger values of θ on some, especially large, instances. In this subsection, we consider two experiments to examine the effect of the table size somewhat more closely. These are summarized in Figure 3.7.

In Figure 3.7 (a), we observe the effect of increasing the maximum allowed table size from our standard setting, which is $\lambda = 10000/\varepsilon$, to $\lambda = 10^6$. One can see that, whereas the improvement is oftentimes not particularly very much (especially remembering that much longer running times have to be accepted, cf. Section 3.3.1–3.3.2), only the cases with large memory demonstrate that increasing the value of θ actually improves the solution.

Figure 3.7 (b) shows what percentage of the 15 instances in each class of the FST preprocessed instances could be solved completely under the given table size limits; in other words, it shows for how many of the instances the given maximum table size was sufficient so that no potential solutions had to be thrown away in the dynamic programming. It is clear in the figure that for $\lambda = 10^6$ considerably more dynamic programs could be solved optimally. But it is also interesting to notice that more instances could be correctly solved for the case $\varepsilon = 0.1, \lambda = 10^5$ than in $\varepsilon = 0.5, \lambda = 20000$. On the other hand, the fact that for the case with 250 terminals and $\lambda = 10^6$, the value for $\varepsilon = 0.1$ is somewhat better than $\varepsilon = 0.5$ can probably be attributed to the presumption that the mortar graph of a particular instance turned out to be more complicated for $\varepsilon = 0.5$ than for $\varepsilon = 0.1$.

3.3.4 The Influence of Treewidth

The most important factor that influences the running time and – since we bound the table size – also the solution quality of the instances is the size of the cuts that separate the subproblems in the dynamic programming. This corresponds, in fact, to the width of the tree decomposition that is given implicitly by the primal tree of the current parcel (see Section 3.2.2– 3.2.3). Note that this is not a tree decomposition of just the mortar graph but that of the brick-contracted graph (see Figure 1.3 (e)). Somewhat imprecisely and for the sake of convenience, we call the maximum size of these cuts, *the treewidth*

of the current instance[4]. In theory, this value is bounded by $2\theta\eta + 1$, where η is the maximum breadth-first-search depth (or outerplanarity) of the parcels. In practice, it is usually somewhat smaller. Figure 3.6 (b) shows the average treewidth of the FST preprocessed instances of SteinLib for various settings of ε and θ. It is interesting and somewhat unexpected to note that on these instances, for fixed θ, the average treewidth is larger for $\varepsilon = 0.5$ than for $\varepsilon = 0.1$. This fact can also clearly be observed by comparing the running times in Table 3.3. Apparently, the structure of the mortar graphs of these instances turns out to be more complicated when constructed with $\varepsilon = 0.5$ than with $\varepsilon = 0.1$.

Figure 3.8 demonstrates the direct exponential dependence of the table size and running time of the algorithm on the treewidth of the instances, regardless of the parameter setting; in other words, the input and the parameters determine the treewidth and the treewidth determines the running time. Note that λ, the maximum allowed table size, is set to one million in this experiment.

3.3.5 Varying the Number of Super-Columns

After the "columns" of each strip are determined (see Section 3.2.1), every κth column is taken to be a "super-column" and is added to the mortar graph. In the theoretical analysis, it is important that the total sum of the weights of the super-columns does not exceed $\mathcal{O}(\varepsilon\,\mathrm{OPT})$ since they are assumed to be added to the final solution. In an implementation, we don't need to add all the super-columns to the solution and we let the dynamic program decide about them. So, it is not absolutely necessary to keep this weight bound; still, we described a strategy for selecting super-columns in Section 3.2.1 that obeys this bound while trying to keep κ as small as possible. In this subsection, we compare this strategy against some fixed values of κ, namely 5, 10, and 20. The results are shown in Table 3.5.

A small value of κ means that we get more and smaller bricks; this can lead to better solutions. On the other hand, since the weight of the super-columns could get too high, we lose the approximation guarantee inside the bricks (but we lose this guarantee anyway because of θ). Indeed, we see in Table 3.5 that we obtain smaller optimality gaps for smaller values of κ. We can also observe that our strategy is similar to the choice of $\kappa = 20$ and, somewhat surprisingly, results in the best average running times. One would expect longer running times for smaller values of κ – because having more bricks should impose a larger treewidth; but this presumption is not completely confirmed by the experiments on these FST preprocessed instances: only for the cases with 1000 terminals, we see that for $\kappa = 5$ we have much longer running times.

3.3.6 Creating More Parcels

The original PTAS has a parameter $\eta = \mathcal{O}(\varepsilon^{-2})$ that specifies how to split the mortar graph into parts of treewidth $\mathcal{O}(\eta)$. It is very important that the total weight of the parcel

[4]The actual treewidth would, of course, be the minimum width minus one over all tree decompositions of the parcel.

boundaries does not exceed $\mathcal{O}(\varepsilon\,\mathrm{OPT})$ since they have to be added to the final solution. In our implementation, like in the case of super-columns, we do not automatically add all parcel boundaries to the solution; we first assume that they are added and solve each parcel separately; then we remove the parcel boundaries and reconnect the solution parts by the strategy explained in Section 3.2.4. So, once again, it is not absolutely necessary that the weight of the parcel boundaries is bounded by $\mathcal{O}(\varepsilon\,\mathrm{OPT})$. We introduced a new parameter γ that specifies the maximum allowed weight of the parcel boundaries to be $\frac{\varepsilon\,\mathrm{OPT}}{\gamma}$ (in fact, we use the lower bound $\frac{\varepsilon\,\mathrm{LB}}{\gamma}$, see Section 3.2.2). In our standard settings, we set $\gamma = 2$. By lowering the value of γ, we might get more parcels and a smaller treewidth for each parcel. We tested the FST preprocessed instances with $\gamma = 2$, 1, and 0.5 (we also tested $\gamma = 1.5$ but the results were either equal to the case of $\gamma = 2$ or $\gamma = 1$). In Table 3.6, we see the results on some instances, in which a change actually occurred.

One can see that the treewidth and hence the running time of the instances drastically reduces if a smaller value of γ can lead to a better partitioning of the mortar graph into parcels. But the optimality gap usually worsens: this is because creating a connected solution out of many parcels, as described in Section 3.2.4, proceeds very similarly to the 2-approximation and can thus cause relatively large errors (the strategy of just adding all parcel boundaries as in the theoretical description would be much worse). Note that sometimes, e.g. the instances esfst1000fst02, esfst1000fst03, and esfst1000fst15, a smaller value of γ can result in a larger or the same treewidth but with a different parcel structure. This is due to the way we split the mortar graph into parcels: as described in Section 3.2.2, we try to split the mortar graph as evenly as possible while obeying the given weight bound on the parcel boundaries; but when we add the bricks, the tree decomposition of the brick-contracted graph can still turn out to have a somewhat larger width.

3.3.7 The LIN Instances from SteinLib

Tables 3.7 and 3.8 show the results of the VLSI derived instances LIN from the SteinLib library. One can see that the batched 1-Steiner heuristic performs extremely well on these instances. Nevertheless, on very large instances it needs a lot of time - sometimes much longer than our PTAS. This is because its running time depends strongly on n whereas our running time depends much more on the treewidth of the instance and the parameter λ. See the next subsection for a further discussion about this matter. Another curious observation is that our algorithm can hardly improve the 1-Steiner solution, when we start with this solution instead of the 2-approximation; in both the FST preprocessed instances (see Section 3.3.1) and the random instances (see Section 3.3.8), we get some, albeit minor, improvements, but on these LIN instances there are almost none or only very minimal improvements. One reason is surely the fact that the 1-Steiner solutions of these instances are almost optimal. But otherwise it seems like the 1-Steiner solution very much imposes its own structure on the dynamic programming; i.e. in order to improve it, some drastically different solution structures have to be found that probably have relatively large weight in the early stages of the dynamic programming and are thus

thrown away due to space limitations. However, this is only a speculation and cannot be firmly backed by data.

3.3.8 Results on Randomly Generated Instances

		Our Solution		1-Steiner		Combined	
n	$\|R\|$	impr.(%)	time (s)	impr.(%)	time (s)	impr.(%)	time (s)
10000	500	2.79	69	3.47	67	3.53	126
50000	500	3.15	148	3.86	1996	4.00	2132
100000	500	3.28	246	3.99	7423	4.03	7651
500000	500	3.05	1506	3.32	234758	3.49	236219
1000000	500	3.71	3668	-	-	-	-

Table 3.2: Results for randomly generated test instances. The table columns show the improvement upon the MST-based 2-approximation algorithm and the computation time in seconds for (1) our PTAS implementation with $\varepsilon = 0.5$, (2) the batched 1-Steiner heuristic, and (3) the combination of these two methods. The values shown are averages over 10 instances in each class. Due to the huge running time, the 1-Steiner and Combined result for the case $n = 500000$ is only computed for one instance.

Results on our randomly generated test instances are shown in Table 3.2. The instances are biconnected planar graphs generated using the Open Graph Drawing Framework (OGDF) [OGD07]. It is remarkable that we can handle instances with up to one million vertices in less than 1.5 hours of computation time. We can see that on these large instances, our PTAS implementation is much faster than the 1-Steiner heuristic while delivering solutions that are qualitatively very close to it. We achieve again slight improvements when combining these two methods. Large instances with a relatively small number of terminals appear often in practice and we observe that we can handle such instances quite well: the reason is that our computation time strongly depends on the size of the mortar graph and when there are few terminals, the mortar graph tends to become small – even in a very large graph. In a sense, the mortar graph/brick-decomposition identifies the most important parts of the graph in the current instance and enables us to concentrate on these parts when searching for near-optimal solutions.

On very large instances, the mortar graph construction time can strongly dominate the dynamic programming time – at least with our standard parameter settings. This fact can clearly be seen in Figure 3.9 (a). The mortar graph construction has a time complexity of $\mathcal{O}(n^2 \log n)$, whereas the dynamic programming runs in time $\mathcal{O}(n\lambda^2)$. In our standard settings, we have $\lambda = 20000$ and this is much smaller than n in this class of instances. Using dynamic trees, one can improve the theoretical running time of the mortar graph construction to $\mathcal{O}(n \log n)$ (see Section 3.2.1) but at the cost of large hidden constants in the $\mathcal{O}$-notation. Implementing and applying dynamic trees to the

mortar graph construction was beyond the scope of this work but would be an interesting experiment for future work.

It is very interesting to observe that the dynamic programming time indeed grows very slowly, even though we are considering instances in the range of ten thousand to one million vertices. This is most likely due to the fact that our instances all have the same number of terminals, namely 500, and so, the mortar graphs do not grow as fast as n. Figure 3.9 (b) shows the average treewidth of the instances and one can see that they are all actually very close to each other. Of course, the fact that λ is constantly equal to 20000 on all tests is also an important factor that causes these small running times.

3.3.9 Summary of Observations

We thoroughly examined the various parameters of the algorithm and in summary, made the following observations. On FST preprocessed instances we could see a consistent improvement in solution quality with smaller values of ε with average optimality gaps of less than 1% for $\varepsilon = 0.1$ (whereas theoretically, this value of ε would only guarantee an optimality gap of 10%). Increasing the number of portals, even slightly, resulted in vastly deteriorating running times while bringing only little improvement; indeed, we believe that setting $\theta = 5$ is an appropriate compromise in solution quality and running time. A main reason why increasing the value of θ does not help very much is that, in general, we cannot afford to increase the table size proportionally (i.e. exponentially); it is only with much larger table sizes that actual improvements are achieved by increasing θ. On the other hand, we observed fairly small improvements by using large table sizes, while having to accept much longer running times. This leads us to the conclusion that increasing the table size by only a constant factor is not sufficient for large improvements, whereas the increase in running time is significant.

We defined the treewidth of an instance as the maximum size of a cut that is considered in the dynamic programming. We observed a direct exponential dependence of the running time on the treewidth and also increasing treewidth values for increasing number of terminals. Still, the values we empirically obtain are much lower than what is predicted in theory.

Our strategy for choosing the value of the parameter κ that determines the selection of super-columns turns out to have a similar effect to choosing rather large values of κ close to 20 while obtaining very good average running times. It seems that smaller values of κ result in somewhat better optimality gaps.

Decreasing the value of γ and thereby attempting to create more parcels seems to be seldom successful. In cases where more parcels are actually created, we often observe a significant improvement in running time in exchange for somewhat worse solution quality.

Our randomly generated instances with a fixed number of terminals equal to 500 turned out to have parcel decompositions with similar treewidths and hence similar running times in the dynamic programming part of the algorithm. The main factor that dominates their running time is indeed the mortar graph construction. We are

able to handle instances with up to one million vertices and produce solutions that are qualitatively very close to the solutions of the 1-Steiner heuristic while our PTAS implementation is up to a factor of 150 faster than the heuristic on large instances.

3.4 Conclusion and Outlook

We provided an implementation of a highly theoretical approximation scheme together with a thorough experimental evaluation of its performance and the influence of its various parameters. We hope that this evaluation helps us understand these parameters more deeply and that it can be a step towards designing better algorithms and approximation schemes for these kind of hard problems on graphs – both in theory and practice. We tried to get to the limit of practicality of algorithms that were generally thought to be "purely theoretical"; and our experiments show that we can already exploit these "theoretical ideas" in practice today and that we might be able to get even better practical fruits from them in the future.

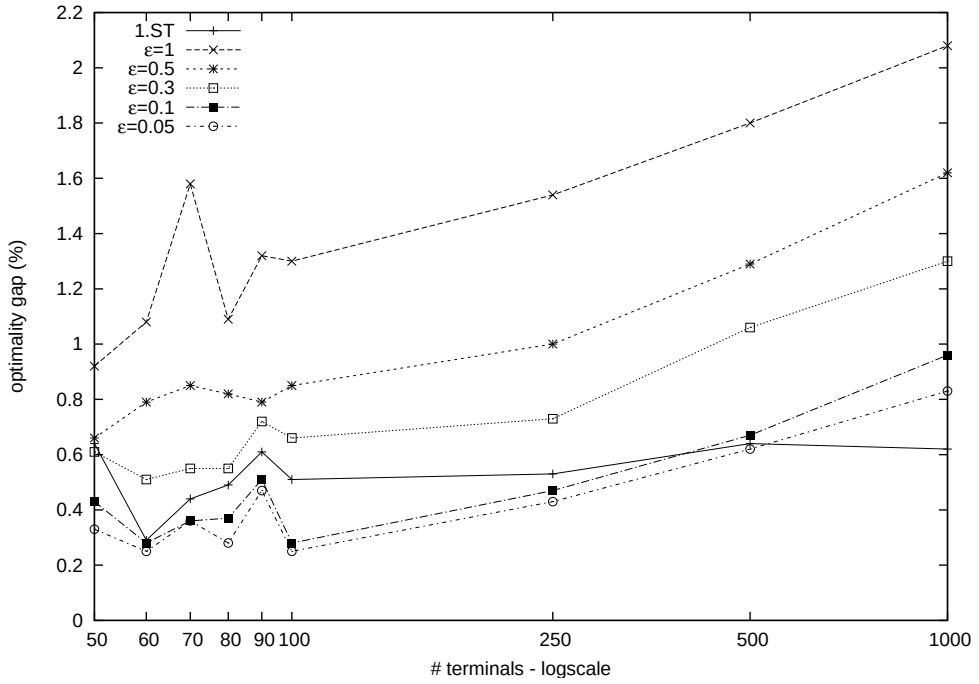
The most interesting technical observation of this work was perhaps the fact that the parameter θ, that specifies the number of portals in each brick, may be chosen to be very small in practice while still delivering excellent results; in theory, this parameter had to be huge in order to provide a rigorous approximation guarantee. Our experiments also suggest that it might be possible to improve the theoretical analysis of the PTAS.

The most important factor that determines the quality and running time of the dynamic programming part of the PTAS is the parameter λ that specifies the maximum allowed table size. In order to improve our implementation, one has to apply some ideas to reduce the required table size in practice. One idea would be to consider each partial solution in the current dynamic programming table, calculate a lower bound on the cost that is needed to complete the solution and throw it away early on, if this lower bound is too large. In very recent experiments, we tried to incorporate the trivial lower bound of taking half of the 2-approximation but it did not bring any noticeable improvements. Incorporating a strong and efficiently computable lower bound for the Steiner tree problem in this PTAS is an important task for future work.

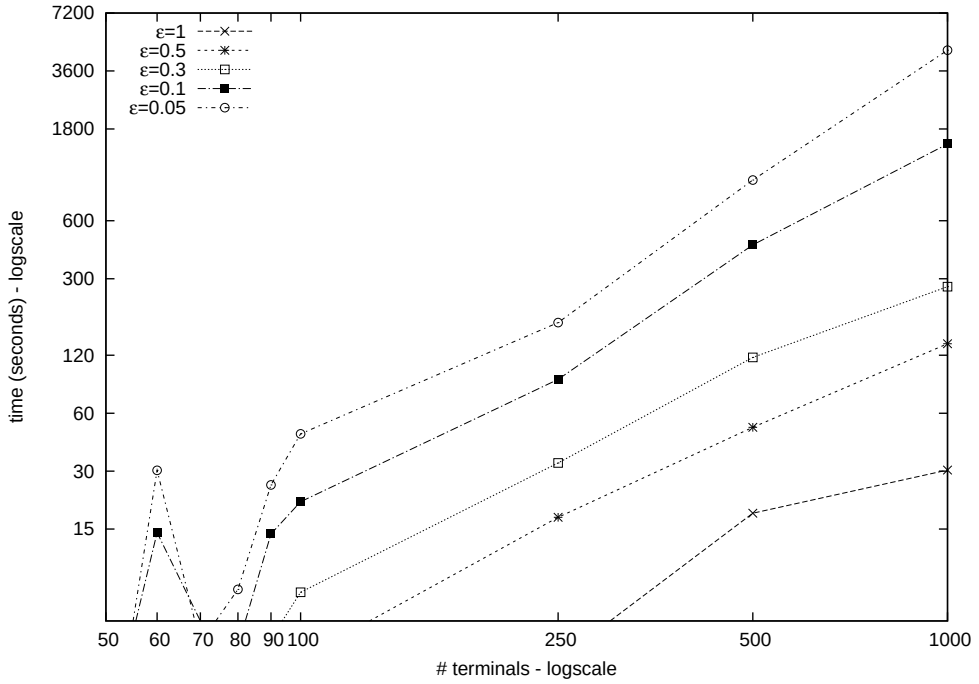
As we saw in Chapter 2, the planar (subset) TSP admits the methodology of the PTAS implemented in this work, too. It would be very interesting to see how the PTAS performs for this problem and especially, if one can drastically reduce the required table sizes using the well-known lower and upper bounds for the TSP.

Interestingly, the structure and the techniques used in many FPT algorithms and EPTASes are similar, especially on planar graphs; hence, it might be possible to attack other FPT algorithms on planar graphs from a practical point of view as well using the techniques introduced in this work.

Finally, on very large instances (such as in Section 3.3.8), we observed that the time needed for the mortar graph construction can dominate the dynamic programming time. For these instances, one could attempt to use dynamic trees such as top trees for the mortar graph construction and see if one can improve the empirical running time.

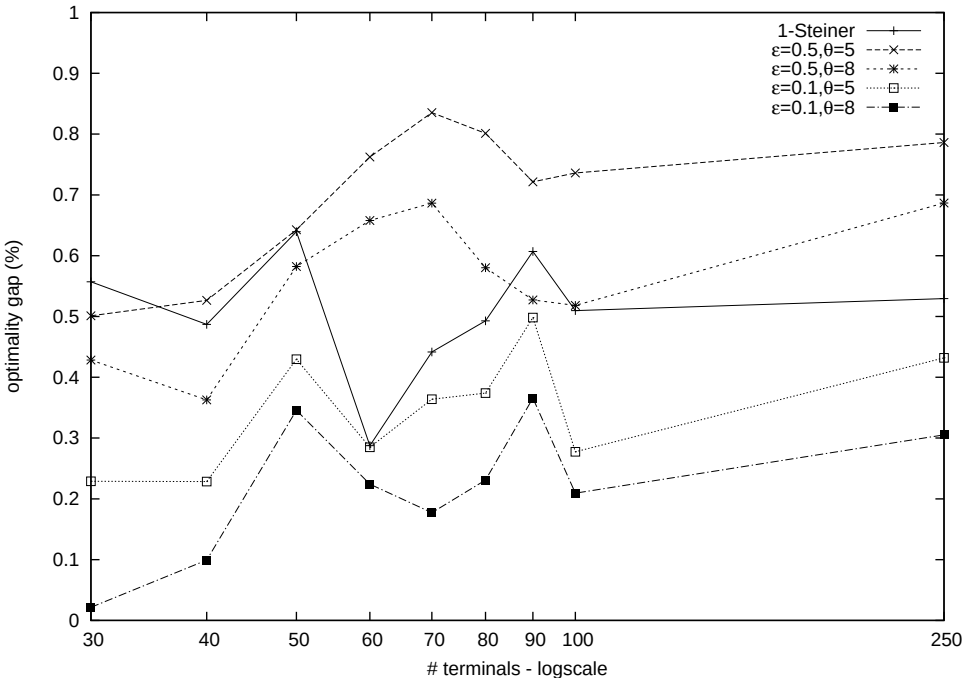


(a)

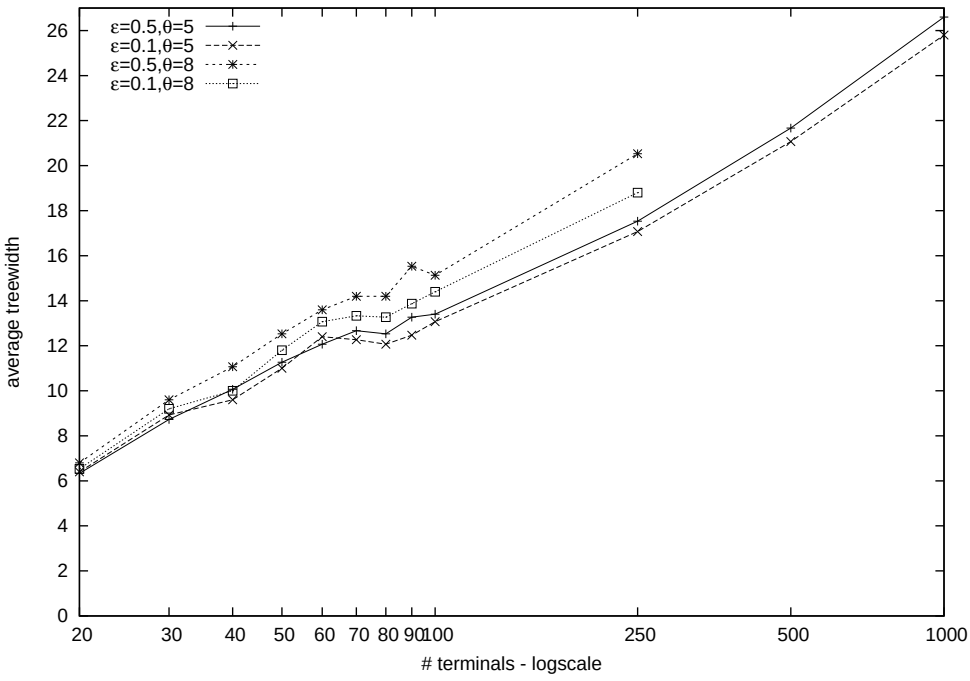


(b)

Figure 3.5: (a) The average optimality gap of FST preprocessed instances of SteinLib for different values of ε and compared to the 1-Steiner heuristic; (b) the average running time of these instances. The averages are taken over 15 instances in each size class.



(a)



(b)

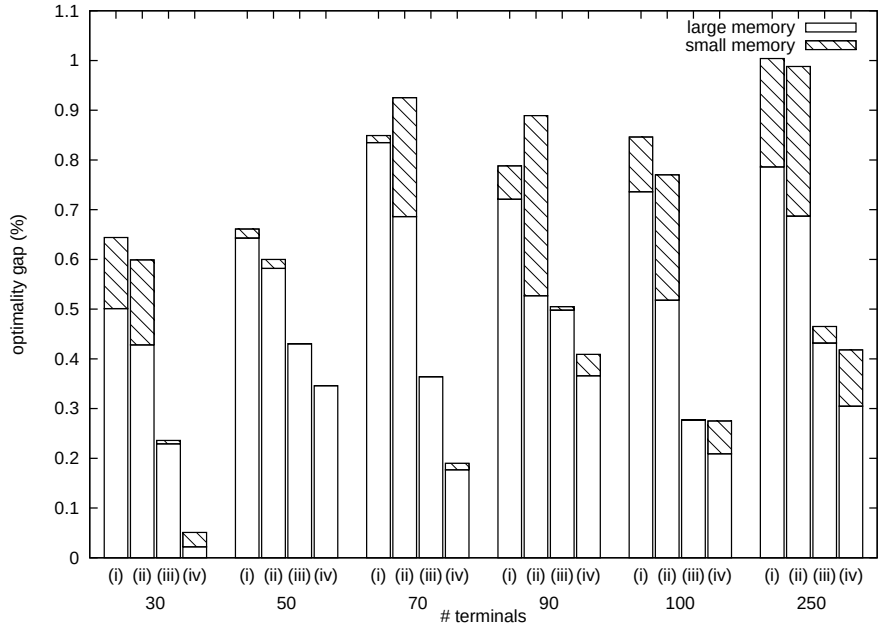
Figure 3.6: (a) The average optimality gap of FST instances of SteinLib for different values of ε and θ and compared to the 1-Steiner heuristic; the maximum allowed table size λ is set to 10^6; (b) the average treewidth of FST preprocessed instances for various parameters.

$\|R\|$	1-St gap(%)	$\varepsilon = 0.5$ $\theta = 5$ gap(%)	$\varepsilon = 0.5$ $\theta = 8$ gap(%)	$\varepsilon = 0.1$ $\theta = 5$ gap(%)	$\varepsilon = 0.1$ $\theta = 8$ gap(%)	$\varepsilon = 0.5$ $\theta = 5$ *time(s)*	$\varepsilon = 0.5$ $\theta = 8$ *time(s)*	$\varepsilon = 0.1$ $\theta = 5$ *time(s)*	$\varepsilon = 0.1$ $\theta = 8$ *time(s)*
10	0.02	0.06	0.06	0.00	0.00	*0.00*	*0.00*	*0.00*	*0.00*
20	0.31	0.20	0.18	0.52	0.42	*0.01*	*0.02*	*0.00*	*0.01*
30	0.56	0.50	0.43	0.23	0.02	*16.26*	*22.50*	*124.69*	*116.75*
40	0.49	0.53	0.36	0.23	0.10	*0.62*	*3.14*	*0.12*	*0.27*
50	0.64	0.64	0.58	0.43	0.35	*3.25*	*10.74*	*1.21*	*3.25*
60	0.29	0.76	0.66	0.28	0.22	*53.25*	*89.44*	*29.89*	*204.86*
70	0.44	0.84	0.69	0.36	0.18	*2.21*	*28.26*	*178.05*	*321.88*
80	0.49	0.80	0.58	0.37	0.23	*6.28*	*49.80*	*25.06*	*44.49*
90	0.61	0.72	0.53	0.50	0.37	*70.25*	*640.78*	*162.20*	*348.37*
100	0.51	0.74	0.52	0.28	0.21	*72.58*	*2562.37*	*62.51*	*341.50*
250	0.53	0.79	0.69	0.43	0.31	*2168.96*	*4522.11*	*1438.03*	*2540.01*
avg.	0.44	0.60	0.48	0.33	0.22	*217.61*	*720.83*	*183.80*	*356.49*

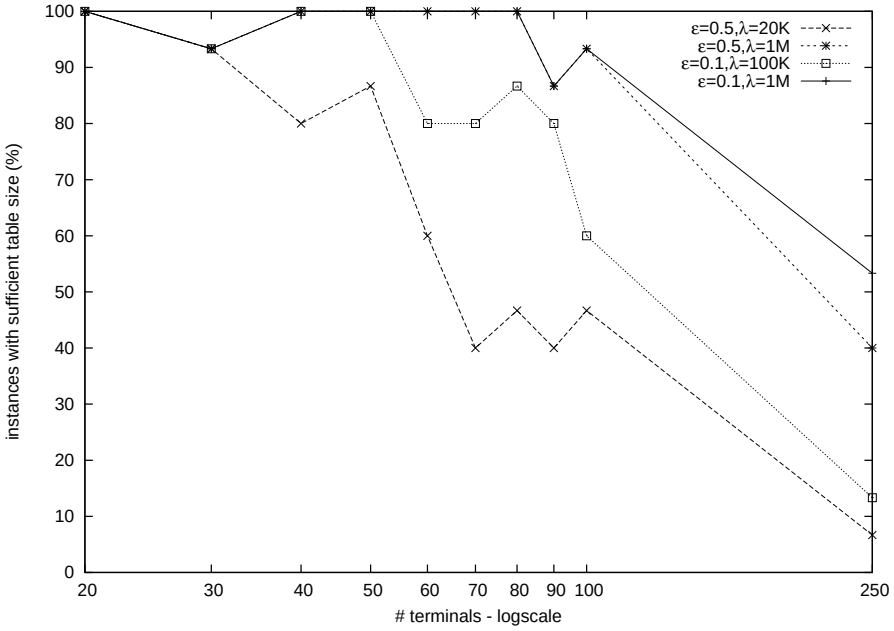
Table 3.3: The average optimality gap and running time of FST instances of SteinLib for different values of ε and θ and compared to the 1-Steiner heuristic; the maximum allowed table size λ is set to 10^6.

#	n	$\varepsilon = 0.5$ $\theta = 5$ gap(%)	$\varepsilon = 0.5$ $\theta = 8$ gap(%)	$\varepsilon = 0.1$ $\theta = 5$ gap(%)	$\varepsilon = 0.1$ $\theta = 8$ gap(%)	$\varepsilon = 0.5$ $\theta = 5$ *time(s)*/**tw**	$\varepsilon = 0.5$ $\theta = 8$ *time(s)*/**tw**	$\varepsilon = 0.1$ $\theta = 5$ *time(s)*/**tw**	$\varepsilon = 0.1$ $\theta = 8$ *time(s)*/**tw**
01	623	1.01	0.62	0.64	0.43	*84.71*/**16**	*7327.20*/**21**	*1007.70*/**17**	*3074.23*/**19**
02	542	0.67	0.56	0.43	0.06	*40.65*/**15**	*311.68*/**19**	*7.94*/**15**	*128.62*/**17**
03	543	0.42	0.49	0.16	0.08	*0.58*/**13**	*5.97*/**14**	*17.51*/**15**	*54.90*/**16**
04	604	0.45	0.45	0.32	0.24	*5824.38*/**21**	*10156.50*/**23**	*484.42*/**17**	*481.02*/**17**
05	596	0.80	0.52	0.33	0.58	*506.18*/**18**	*4602.81*/**23**	*1317.10*/**20**	*3169.20*/**21**
06	596	0.86	1.02	0.38	0.18	*2153.92*/**21**	*11467.00*/**25**	*468.47*/**16**	*694.24*/**18**
07	585	0.62	0.55	0.24	0.39	*85.93*/**16**	*965.86*/**17**	*162.98*/**16**	*271.44*/**17**
08	657	1.22	1.53	0.70	0.29	*674.04*/**20**	*2272.06*/**23**	*54.03*/**16**	*379.81*/**17**
09	570	0.75	0.72	0.31	0.30	*15.89*/**15**	*1366.40*/**18**	*14.63*/**15**	*136.62*/**16**
10	662	0.93	0.64	0.46	0.26	*251.21*/**18**	*1339.50*/**20**	*270.65*/**16**	*1519.99*/**19**
11	661	0.78	0.37	0.35	0.13	*466.30*/**18**	*2729.73*/**20**	*64.73*/**15**	*190.39*/**16**
12	619	0.58	0.51	0.25	0.02	*363.93*/**16**	*2539.38*/**20**	*777.88*/**17**	*4878.98*/**19**
13	684	0.64	0.38	0.31	0.13	*16391.40*/**19**	*8726.67*/**23**	*754.34*/**19**	*929.25*/**21**
14	710	1.27	1.17	1.09	0.92	*4202.02*/**19**	*8826.75*/**22**	*5478.85*/**21**	*15152.20*/**23**
15	713	0.78	0.77	0.52	0.58	*1473.27*/**18**	*5194.14*/**20**	*10689.20*/**21**	*7039.26*/**26**

Table 3.4: The detailed results of Section 3.3.2 for the FST preprocessed instances with 250 terminals. The first column gives the instance number in SteinLib. The bold numbers in the columns labeled **tw** show the resulting treewidth of the instance. The maximum allowed table size is set to 10^6.

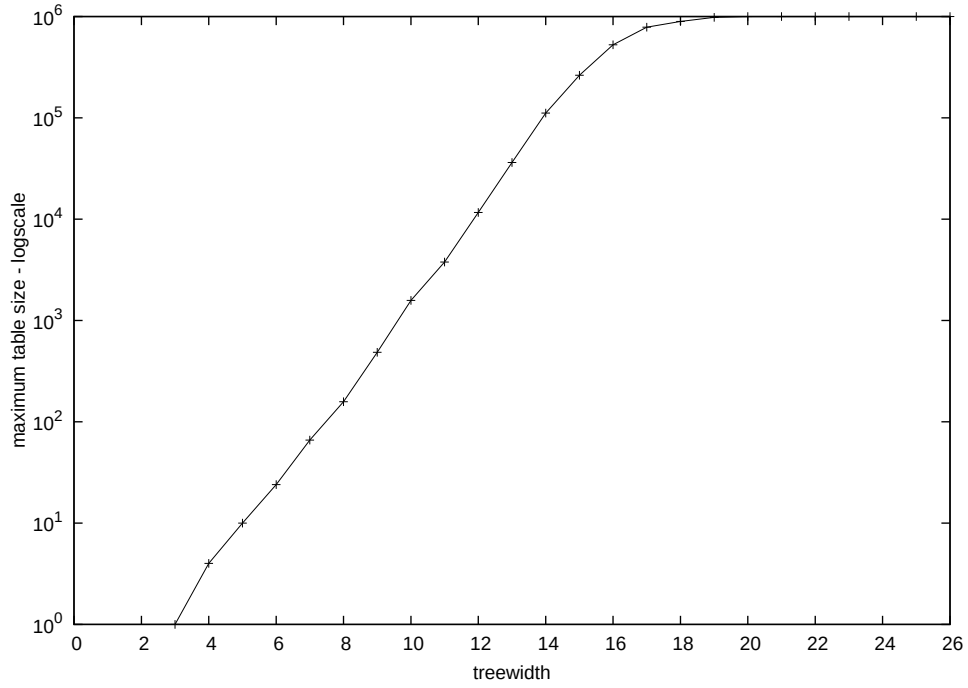


(a)

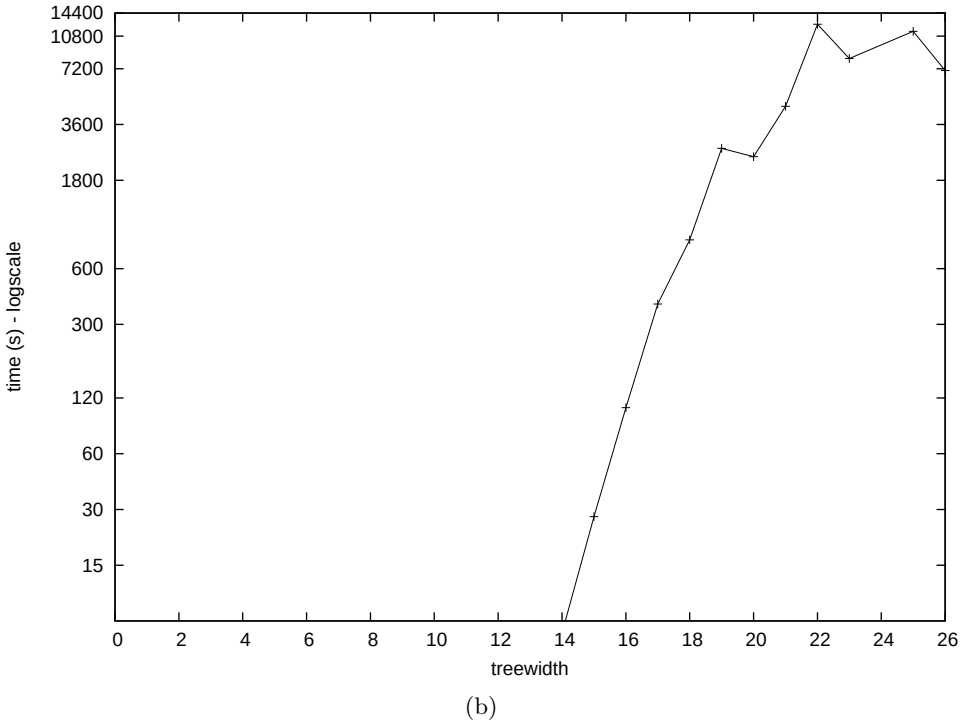


(b)

Figure 3.7: (a) The effect of increasing the memory in the cases (i) $\varepsilon = 0.5$, $\theta = 5$ (ii) $\varepsilon = 0.5$, $\theta = 8$ (iii) $\varepsilon = 0.1$, $\theta = 5$ (iv) $\varepsilon = 0.1$, $\theta = 8$; the empty bars show the optimality gap for $\lambda = 1$M and the filled bars the increase in the gap when using our standard settings of $\lambda = 20$K and 100K, respectively. (b) The percentage of FST preprocessed instances for which the table size was sufficient for the shown parameter settings (with $\theta = 5$, 1M $= 10^6$, 1K $= 10^3$).



(a)



(b)

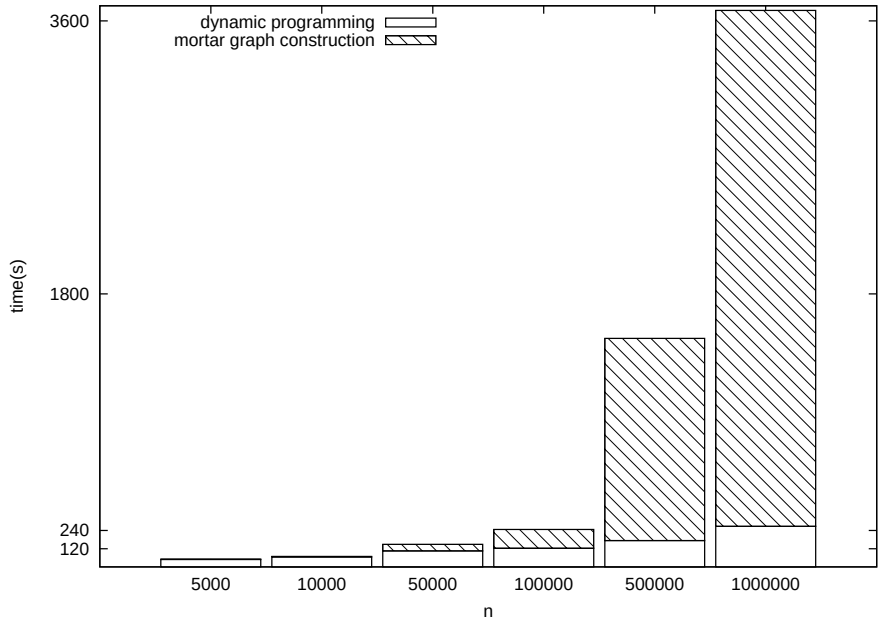
Figure 3.8: The average (a) maximum table size and (b) running time of FST preprocessed instances relative to their treewidth. The averages are taken over all FST preprocessed instances with up to 250 terminals, $\lambda = 10^6$, and all combinations of $\varepsilon = 0.5, 0.1$ and $\theta = 5, 8$.

$\|R\|$	Our Strat. gap(%)	$\kappa = 5$ gap(%)	$\kappa = 10$ gap(%)	$\kappa = 20$ gap(%)	Our Strat. *time(s)*	$\kappa = 5$ *time(s)*	$\kappa = 10$ *time(s)*	$\kappa = 20$ *time(s)*
50	0.66	0.48	0.58	0.66	*0.95*	*0.57*	*0.66*	*1.03*
100	0.85	0.63	0.72	0.85	*3.24*	*4.58*	*3.55*	*3.51*
250	1.00	0.78	0.91	0.96	*17.27*	*14.86*	*18.35*	*18.47*
500	1.29	1.22	1.24	1.29	*50.73*	*49.85*	*53.72*	*56.48*
1000	1.62	1.40	1.51	1.64	*137.76*	*154.19*	*143.82*	*149.14*
avg.	1.08	0.90	0.99	1.08	*41.99*	*44.81*	*44.02*	*45.72*

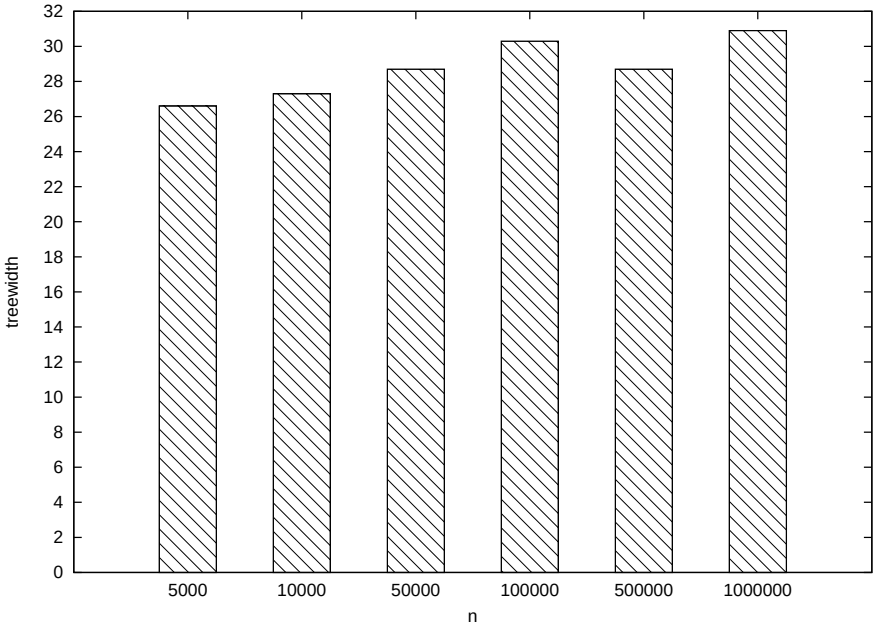
Table 3.5: Results on FST preprocessed instances with different values of κ. The tests are done with otherwise standard settings, in particular $\varepsilon = 0.5, \theta = 5, \lambda = 20000$.

Instance	n	$\gamma = 2$			$\gamma = 1$			$\gamma = 0.5$		
		tw/#p	gap(%)	*time(s)*	**tw**/#p	gap(%)	*time(s)*	**tw**/#p	gap(%)	*time(s)*
es100fst01	250	**14**/1	1.07	*4.84*	**13**/2	1.51	*0.30*	**13**/2	1.51	*0.29*
es100fst07	276	**16**/1	0.11	*8.61*	**16**/1	0.11	*8.65*	**14**/2	1.26	*0.75*
es250fst06	596	**21**/1	1.14	*55.60*	**14**/7	0.90	*0.30*	**14**/7	0.90	*0.30*
es250fst13	684	**19**/1	0.64	*22.77*	**19**/1	0.64	*23.05*	**15**/12	1.79	*1.54*
es500fst05	1172	**21**/1	0.83	*27.58*	**21**/1	0.83	*27.92*	**13**/11	1.42	*1.15*
es500fst12	1322	**27**/1	1.48	*113.21*	**20**/3	1.23	*30.02*	**20**/3	1.23	*30.09*
es1000fst02	2629	**29**/1	1.26	*112.29*	**21**/25	1.77	*96.39*	**22**/9	1.74	*97.08*
es1000fst03	2762	**28**/1	1.81	*162.53*	**21**/18	1.65	*113.07*	**24**/19	2.28	*52.32*
es1000fst09	2846	**30**/1	2.08	*203.97*	**18**/8	1.66	*40.70*	**18**/8	1.66	*40.78*
es1000fst15	2733	**28**/1	1.59	*181.53*	**19**/18	1.74	*84.48*	**19**/10	1.60	*62.30*
average		**23**/1	1.20	*89.29*	**18**/8	1.20	*42.49*	**17**/8	1.54	*28.66*

Table 3.6: Results on some FST preprocessed instances with different values of γ and otherwise standard settings. The number of terminals can be read from the instance name. The columns in bold show the resulting treewidth of the instances and the columns labeled #p indicate the number of parcels.



(a)



(b)

Figure 3.9: (a) The average running time of our algorithm on the randomly generated instances split into the dynamic programming time and the mortar graph construction time. (b) The average treewidth of the our randomly generated instances. Recall that we consider tree decompositions of the parcels, which are subgraphs of the brick-contracted graphs; hence, their treewidth is much more influenced by the number of terminals (and the parameters) than the total number of vertices.

	n	m	t	OPT	2-OPT	Our Sol.	1-St	Comb.	Our Sol. *time(s)*	1-St *time(s)*	Combined *time(s)*
lin01	53	80	4	**503**	503	503	503	503	*0*	*0*	*0*
lin02	55	82	6	**557**	557	557	557	557	*0*	*0*	*0*
lin03	57	84	8	**926**	932	926	926	926	*0*	*0*	*0*
lin04	157	266	6	**1239**	1267	1239	1239	1239	*0*	*0*	*0.01*
lin05	160	269	9	**1703**	1808	1703	1703	1703	*0*	*0.01*	*0.01*
lin06	165	274	14	**1348**	1422	1356	1348	1348	*0.04*	*0.01*	*0.03*
lin07	307	526	6	**1885**	2007	1911	1885	1885	*0.01*	*0.02*	*0.18*
lin08	311	530	10	**2248**	2308	2284	2248	2248	*0.01*	*0.03*	*0.03*
lin09	313	532	12	**2752**	2785	2785	2752	2752	*1.21*	*0.02*	*0.64*
lin10	321	540	20	**4132**	4400	4230	4169	4149	*0.27*	*0.03*	*2.11*
lin11	816	1460	10	**4280**	4335	4287	4287	4287	*1.67*	*0.14*	*0.21*
lin12	818	1462	12	**5250**	5356	5301	5301	5301	*0.12*	*0.16*	*0.53*
lin13	822	1466	16	**4609**	4850	4620	4618	4618	*1.41*	*0.16*	*2.68*
lin14	828	1472	22	**5824**	6163	6132	5830	5830	*1.69*	*0.36*	*1.36*
lin15	840	1484	34	**7145**	7412	7288	7169	7169	*15.65*	*0.2*	*20.08*
lin16	1981	3633	12	**6618**	6806	6759	6618	6618	*0.08*	*0.78*	*0.84*
lin17	1989	3641	20	**8405**	9468	8946	8405	8405	*14.68*	*0.9*	*13.11*
lin18	1994	3646	25	**9714**	10790	10071	9720	9720	*17.07*	*1.35*	*5.73*

Table 3.7: Results on the LIN instances from SteinLib with standard settings and in comparison with the 1-Steiner heuristic (Part I). The columns labeled 'Combined' show the results of starting the PTAS with the 1-Steiner solution.

	n	m	t	OPT	2-OPT	Our Sol.	1-St	Comb.	Our Sol. *time(s)*	1-St *time(s)*	Comb. *time(s)*
lin19	2010	3662	41	**13268**	13942	13678	13268	13268	*21.92*	*1.5*	*24.3*
lin20	3675	6709	11	**6673**	7199	7142	6673	6673	*10.17*	*4.05*	*5.6*
lin21	3683	6717	20	**9143**	9646	9522	9222	9191	*15.71*	*2.75*	*16.44*
lin22	3692	6726	28	**10519**	11095	10712	10545	10545	*7.49*	*2.77*	*6.61*
lin23	3716	6750	52	**17560**	18507	17889	17591	17591	*29.61*	*4.76*	*57.85*
lin24	7998	14734	16	**15076**	16669	16078	15474	15474	*31.38*	*18.55*	*37.23*
lin25	8007	14743	24	**17803**	19457	18617	17821	17821	*18.59*	*18.47*	*38.24*
lin26	8013	14749	30	**21757**	23751	22969	21820	21768	*18.53*	*18.92*	*30.13*
lin27	8017	14753	36	**20678**	22502	21764	20719	20719	*33.3*	*25.14*	*56.46*
lin28	8062	14798	81	**32584**	34452	33915	32736	32718	*80.51*	*21.14*	*186.94*
lin29	19083	35636	24	**23765**	25631	24716	23829	23829	*22.14*	*166.66*	*173.37*
lin30	19091	35644	31	**27684**	30492	29685	27808	27808	*36.96*	*127.04*	*154.94*
lin31	19100	35653	40	**31696**	34846	33442	31770	31770	*37.7*	*131.02*	*163.38*
lin32	19112	35665	53	**39832**	43055	41837	40022	40022	*136.53*	*135.18*	*192.25*
lin33	19177	35730	117	**56061**	59732	57937	56212	56212	*134.06*	*151.84*	*246.44*
lin34	38282	71521	34	**45018**	49912	48646	45596	45548	*44.04*	*1070.93*	*1124.92*
lin35	38294	71533	45	**50559**	54584	52846	50762	50762	*119.57*	*674.17*	*760.08*
lin36	38307	71546	58	**55608**	61549	58427	56294	56294	*115.9*	*671.09*	*761.34*
lin37	38418	71657	172	**99560**	107681	103764	99720	99688	*294.71*	*944.8*	*1231.43*

Table 3.8: Results on the LIN instances from SteinLib with standard settings and in comparison with the 1-Steiner heuristic (Part II). The columns labeled 'Combined' show the results of starting the PTAS with the 1-Steiner solution.

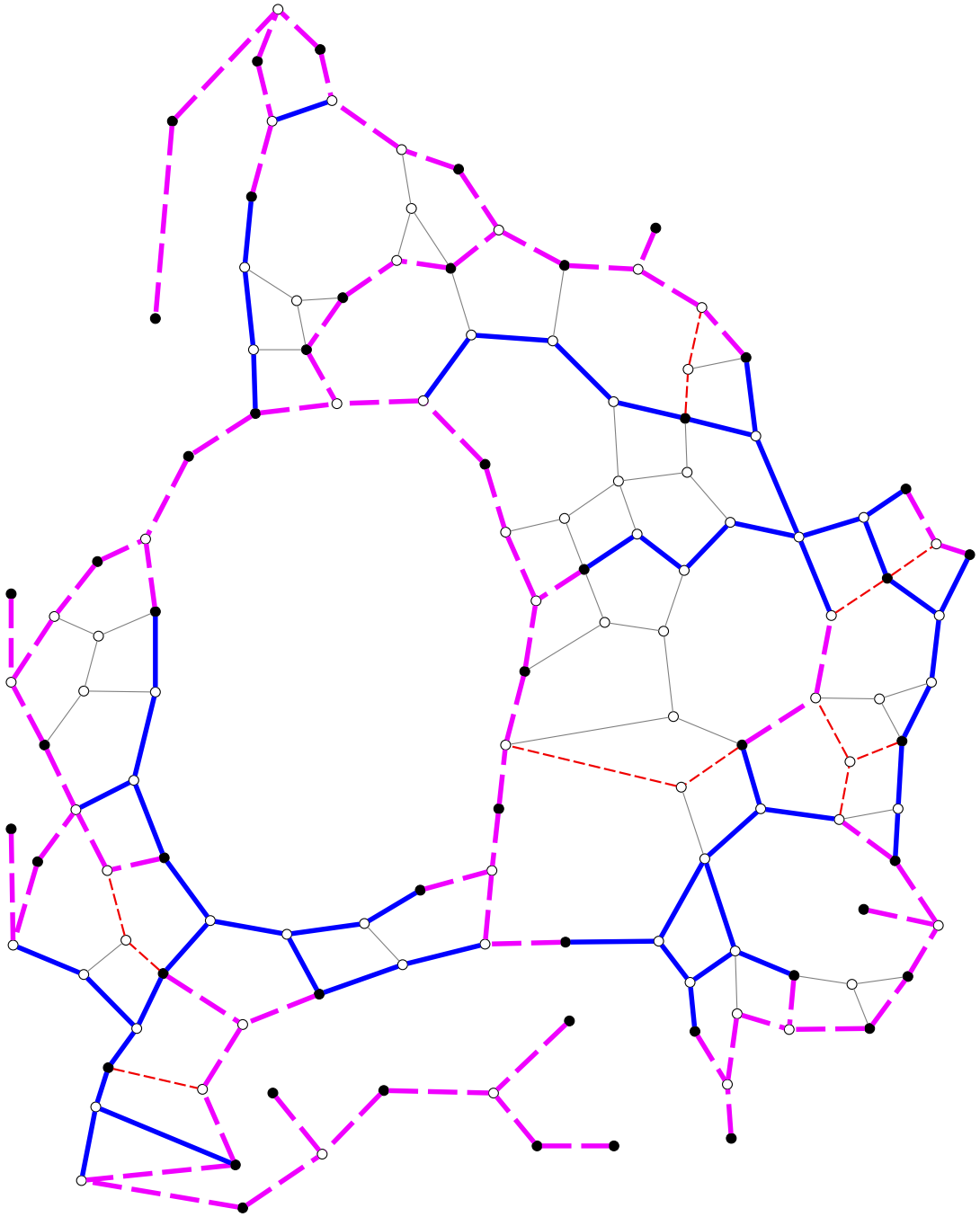


Figure 3.10: An unweighted graph on 128 vertices, 182 edges, and 50 black terminals. A mortar graph with $\varepsilon = 0.8$ is indicated with thick blue and purple lines. A Steiner tree that is calculated with our implementation with $\theta = 8$ and $\lambda = 10^6$ is drawn with dashed red and purple lines. The thick dashed purple lines are the edges that are shared between the mortar graph and the Steiner tree. Notice the bricks that actually contain parts of the solution, i.e. the thin dashed red lines. This image was created using the OGDF library [OGD07] and the yEd graph editor [yEd10].

4 An $\mathcal{O}(n \log^2 n)$ PTAS for Steiner tree among Obstacles in the Plane[1]

We consider the following network design problem: given a set of points in the plane and a set of disjoint polygonal obstacles, find the shortest network interconnecting the points and avoiding the interior of the obstacles. We refer to the given points as *terminals* and to the obstacle vertices as *corners* and assume that no terminal is placed in the interior of an obstacle. We let n be the total number of terminals *and* corners. The shortest interconnecting network of the terminals will be a tree, a *Steiner tree*, and it might use corners and additional vertices called *Steiner points*. Note that we use this term only to refer to points that do not coincide with terminals or corners. This problem is called the *obstacle-avoiding Steiner minimum tree* problem (SMTO) or ESMTO when we want to emphasize we are using the Euclidean metric (see Fig. 4.1(a)).

Uniform orientation metrics are derived from *λ-geometries*. In a λ-geometry, one is allowed to move only along $\lambda \geq 2$ orientations building consecutive angles of π/λ. The *rectilinear* or *Manhattan* metric corresponds to the 2-geometry and the *octilinear* metric to the 4-geometry. We call the corresponding SMT problems λ-SMT or, when obstacles are to be avoided, λ-SMTO. In this case, the obstacle edges must obey the restrictions of the given orientations, too (see Fig. 4.1(b)).

It has been a long-standing open problem whether these SMT problems *among obstacles* admit a polynomial-time approximation scheme [Pro88, LZSK02, MS06]. With the recent result of Borradaile et al. [BKM09] about Steiner trees in planar graphs, this question can now be answered affirmatively for the Euclidean, rectilinear, and octilinear metrics by combining a number of results in the literature (see below). However, the resulting time complexity is $\mathcal{O}(n^2 \log n)$ whereas in order to obtain a near linear running time, new ideas and more sophisticated techniques are required; this is the main contribution of this chapter. Also, we obtain our result for all uniform orientation metrics. Our approach is based on constructing a planar graph of size $\mathcal{O}(n \log n)$ that contains a $(1 + \varepsilon)$-approximation of the solution and then finding an approximate solution in that graph. The total running time will be $\mathcal{O}(n \log^2 n)$. We widely use the notions of *spanners* and *banyans* in our work (see below): informally, a spanner is a subgraph of a given graph that preserves approximate shortest paths while having low total weight; a banyan additionally preserves approximate Steiner trees. Along the way, we prove a number of spanner results and other properties of SMTOs both for the Euclidean and uniformly oriented case.

Recall from Section 1.1.2 that the SMT problem and its many variations are also of high practical relevance [FG82, HRW92, KR95, CKM$^+$98, CD01]. Especially the

[1] This chapter is based on joint work with Matthias Müller-Hannemann [MT07, MT10].

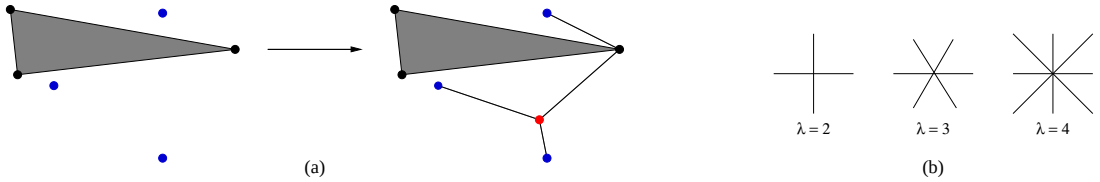


Figure 4.1: (a) An example of SMTO with 3 terminals and an obstacle with 3 corners, i.e. $n = 6$; on the right we see a solution using 1 Steiner point; (b) examples of λ-geometries.

geometric case with obstacles is very important in VLSI design, since there are usually regions in the plane that may not be crossed by wire [GC94, ZW99]. Also, it is often only allowed to route the tree along a rectilinear or octilinear grid and so, SMTOs in uniformly oriented metrics are required [Tei02, CCK+03, KMZ03, PWZ04, MS06].

Related Work

The ESMTO problem is clearly NP-hard since it contains the Steiner minimum tree problem without obstacles as a special case [GGJ77]. For the SMT problem without obstacles, Arora [Aro98] and Mitchell [Mit99] were the first to present a PTAS. Rao and Smith [RS98] improved the running time of Arora's algorithm from $\mathcal{O}(n(\frac{1}{\varepsilon}\log n)^{\mathcal{O}(1/\varepsilon)})$ to $\mathcal{O}(2^{\text{poly}(1/\varepsilon)}n + n\log n)$ using banyans and this is the best running time known so far. All these algorithms require a so-called "patching lemma", basically saying that the part of a solution that lies inside a given rectangle in the plane can be replaced by a part that is not much longer, in such a way that it crosses the boundary of the rectangle at at most a constant number of points. But in the presence of a large number of obstacles inside a rectangle, any solution might be forced to cross the boundary of the rectangle at a large number of points. Hence, these techniques can not be applied directly to the case with obstacles.

Provan [Pro88] has shown how to approximate ESMTO by an SMT problem in graphs and derived an FPTAS for the special case when the terminals lie on a constant number of "boundary polygons" and interior points. The first exact algorithm for ESMTO is given by Zachariasen and Winter [ZW99]. Note however that one has to be careful with "exactness" since computing the length of an ESMT(O) requires the calculation of square roots and thus, this problem is not even known to be in NP. But for the purpose of approximation algorithms, this is not a problem since one can round the results to the required precision.

The PTASes discussed above also apply to λ-SMTs for all $\lambda \geq 2$. The rectilinear and octilinear case have been shown to be NP-complete in [GJ77, MS07]. For general fixed λ no proof has been published so far, though it is widely believed that these problems are hard, too. Properties of uniformly oriented SMTs have been studied by Brazil et al. [BTW00] and exact algorithms have been proposed by Nielsen et al. [NWZ02]. Approximation algorithms for rectilinear SMTO have been proposed by Ganley and Cohoon [GC94] and

for the octilinear case by Müller-Hannemann and Schulze [MS06, MS07]. For rectilinear SMTO with a constant number of obstacles, Liu et al. [LZSK02] presented a PTAS based on Mitchell's [Mit99] approach. The SMT problem with length restrictions on obstacles has been studied by Müller-Hannemann and Peyer [MP03] in the rectilinear case, and by Müller-Hannemann and Schulze [MS06] in the octilinear case, and constant-factor approximation algorithms have been proposed.

As we saw in Chapter 1, the Steiner tree problem in planar graphs has very recently been shown to admit a PTAS by Borradaile et al. [BKM09]. The running time of the PTAS is $\mathcal{O}(n \log n)$ with a constant that is singly exponential in a polynomial in $1/\varepsilon$. This immediately implies a PTAS for rectilinear, octilinear, and Euclidean SMTO by reducing these problems to the planar graph case using the following results from the literature: the so-called Hanan-grid [GC94, Han66] for the rectilinear case, the result of Müller-Hannemann and Schulze [MS06] for the octilinear case, and Provan's construction [Pro88] together with the planar spanner result of Arikati et al. [ACC$^+$96] for the Euclidean case. However, in all these cases, the PTAS of Borradaile et al. has to be run on a graph of size $\mathcal{O}(n^2)$ and thus, the total running time will be $\mathcal{O}(n^2 \log n)$. In this work, we show alternative constructions with running time $\mathcal{O}(n \log^2 n)$. We also briefly discuss some methods that could possibly result in a reduction of the running time to $\mathcal{O}(n \log n)$.

On Spanners and Banyans

The *visibility graph* of a set of terminals and obstacles in the plane is the graph that contains all straight-line connections between terminals and corners that do not cross the interior of any obstacle (see Fig. 4.2(a)). A *t-spanner* of a set of points P is a graph that contains a path between any two points of P that is at most a factor of t longer than the shortest path between them. Spanners have been vastly studied in the literature [Epp00b] and have often been used in the design of PTASes [RS98]. Of particular interest to us are spanners of the visibility graph among obstacles in the plane (see Fig. 4.2(b)). Clarkson [Cla87] showed how to construct a $(1+\varepsilon)$-spanner of linear size of the visibility graph in time $\mathcal{O}(n \log n)$. A linear-sized *planar* spanner for both the rectilinear and Euclidean metric has been shown to exist and to be computable in $\mathcal{O}(n \log n)$ time by Arikati et al. [ACC$^+$96, Zeh02]. We show how to extend these ideas to derive sparse planar spanners for all uniform orientation metrics within the same time.

Rao and Smith [RS98] introduced the notion of *banyans*. A banyan is a graph that contains a $(1+\varepsilon)$-approximation of the SMT of a given set of points and whose weight is at most a constant factor larger than the SMT. Rao and Smith showed how to construct a banyan of size $\mathcal{O}(n)$ in time $\mathcal{O}(n \log n)$ in the obstacle-free case.[2]

[2]Their construction was in fact more powerful as it included an approximate SMT for *any* subset of the terminals.

Figure 4.2: (a) The visibility graph of a given point set with obstacles and (b) a spanner thereof.

Contribution and Outline of this Chapter

The main result of this chapter is the following theorem:

Theorem 4.1. *The Steiner minimum tree problem among disjoint polygonal obstacles in the plane admits a* PTAS *in the Euclidean metric and in all uniform orientation metrics. The running time is $\mathcal{O}(n \log^2 n)$, where n is the total number of terminals and obstacle corners.*

In order to prove Theorem 4.1, we show how to construct a *planar banyan for* SMTO of size $\mathcal{O}(n \log n)$ in $\mathcal{O}(n \log^2 n)$ time by building on the framework of Rao and Smith using new ideas and combining other results from the literature, especially the banyan-result for planar graphs contained in the work of Borradaile et al. [BKM09]. An approximate Steiner tree can then be obtained on this planar graph using [BKM09]. Since the algorithm in [BKM09] is exponential in $1/\varepsilon$, so is our resulting algorithm.

The main difficulties that arise when obstacles are present are to deal with visibility and the fact that only a subset of corners is included in the SMT, i.e. we do not know which vertices of a spanner will be part of the SMT. In particular, a spanner might include arbitrarily short edges between corners that are not part of the SMT and this causes an important proof idea of Rao and Smith to fail. Roughly speaking, they show that in the obstacle-free case, there is always a "long enough" spanner edge near non-negligible SMT edges and so, they introduce a grid of candidate Steiner points in a neighborhood around every spanner edge to capture these SMT vertices. Our main new algorithmic idea is to use $\mathcal{O}(\log n)$ layers of candidate Steiner points around each spanner edge, so that we are guaranteed to find such appropriate points even when our spanner edges are short. Another important difference is that we use planar spanners, so that afterwards, we can use the algorithm of [BKM09] instead of building on Arora's approach [Aro98] to obtain our PTAS. We present our algorithm in Section 4.1 and then present two proofs for the correctness of our algorithm for the Euclidean case in Section 4.2: one using an analog of the so-called hexagon property [GGJ77] and another one using a generalization of the empty ball lemma [RS98]. Even though our proofs follow the lines of the proofs of Rao and Smith, they differ conceptually at some key points and other techniques have to be used (see Section 4.2.1).

Afterwards, in Section 4.3, we turn our attention to uniform orientation metrics and argue how the presented proofs can be modified to work for these cases, too. In Sec-

tion 4.4, we prove a variation of Arikati et al.'s planar spanner result [ACC+96] to apply to uniform orientation metrics. In Section 4.5 we conclude with an outlook on how to possibly reduce the running time to $\mathcal{O}(n \log n)$.

4.1 The Algorithm

Our algorithm is summarized as Algorithm SMTO-PTAS below. We are given a set of terminals Z and a set of disjoint polygonal obstacles O as described in the introduction. Let n be the total number of terminals and obstacle corners. In the first step, we find a $(1+\varepsilon_1)$-spanner G_1 of the visibility graph of $Z \cup O$ using the algorithm of Clarkson [Cla87]. We argue in Section 4.4, that this algorithm also applies to all uniform orientation metrics. The spanner G_1 has n vertices and $\mathcal{O}(n)$ edges and can be found in time $\mathcal{O}(n \log n)$. Note that it is not needed to explicitly construct the full visibility graph.

Algorithm SMTO-PTAS(Z, O, ε)**.**
Input. a set of terminals Z, a set of disjoint polygonal obstacles O in the plane, and the desired accuracy $0 < \varepsilon \leq 1$
Output. a $(1+\varepsilon)$-approximation of the SMTO of the terminals
Note. κ is a constant and can be ≤ 226 in the Euclidean and ≤ 50 in the rectilinear case, ε_1 and ε_2 have to be chosen appropriately, e.g. $\varepsilon_1 = \frac{\varepsilon}{52}$ and $\varepsilon_2 = \frac{\varepsilon}{6}$.

1. find a $(1+\varepsilon_1)$-spanner G_1 of the visibility graph of $Z \cup O$;
2. let $P_0 = \varnothing$;
3. for each edge $e \in E(G_1)$ and $i = 0, \ldots, \lceil \log_2 n \rceil$:
   - let $\ell := \ell(e)$ and $r := \kappa 2^i \ell / \varepsilon_1$;
   - let C be a circle of radius r around the midpoint of e;
   - place a grid with spacing $\delta(r) :== r\varepsilon_1^3/\kappa^2$ inside C; (the grid has $\leq 4\kappa^4/\varepsilon_1^6 = \mathcal{O}(1)$ points)
   - add these points to P_0;
4. remove all the points from P_0 that lie inside obstacles; (let G_2 be the visibility graph of $Z \cup P_0 \cup O$)
5. find a planar $(1+\varepsilon_2)$-spanner G_3 of G_2;
6. find a $(1+\varepsilon/3)$-approximate SMT T of Z in G_3; (using the PTAS of Borradaile et al. [BKM09])
7. return T;

Around each edge of G_1, we consider $\lceil \log_2 n \rceil$ circles with doubling radii and place a grid of constant size inside each of them. This introduces a set P_0 of $\mathcal{O}(n \log n)$ "candidate Steiner points"(see Fig. 4.3). Here, we make use of a constant κ that depends on the metric being used. For $\varepsilon_1 \leq 1$, in the Euclidean metric, κ can be chosen to be ≤ 226, and in the rectilinear metric ≤ 50. Let $k = n + |P_0| = \mathcal{O}(n \log n)$ be the total number of terminals, obstacle corners, and candidate Steiner points. We remove the

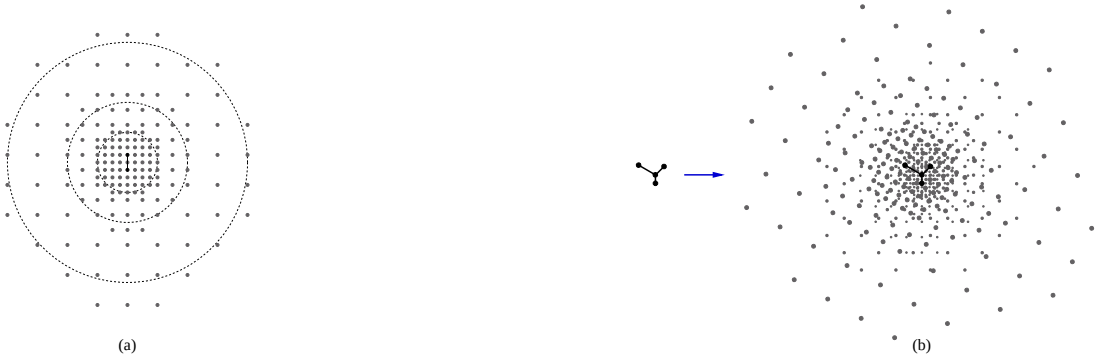


Figure 4.3: (a) Adding candidate Steiner points around one edge of the spanner; (b) and around three edges.

ones that lie inside obstacles using a sweep-line algorithm [SH76] in time $\mathcal{O}(k \log k) = \mathcal{O}(n \log^2 n)$.

Let G_2 be the visibility graph of $Z \cup P_0 \cup O$ and let G_3 be a *planar* $(1+\varepsilon_2)$-spanner of G_2. G_3 can be found using Arikati et al.'s algorithm [ACC+96] or our extension of it for other uniform orientation metrics (see Section 4.4). These spanner algorithms need $\mathcal{O}(k \log k)$ time and introduce $\mathcal{O}(k)$ additional Steiner points to achieve planarity. Thus, G_3 can be constructed in $\mathcal{O}(n \log^2 n)$ time and has $\mathcal{O}(n \log n)$ vertices and edges (note again that G_2 need not be constructed explicitly). Now we find a $(1 + \varepsilon/3)$-approximate Steiner minimum tree of the terminals Z in G_3 using the PTAS of Borradaile et al. [BKM09] for the Steiner tree problem in planar graphs. The time needed for this step is $\mathcal{O}(k \log k)$ and hence, the total runtime of our algorithm is $\mathcal{O}(n \log^2 n)$.

Note that the first step of the PTAS of Borradaile et al. is to determine a subgraph G_4 of G_3 that contains a $(1 + \varepsilon/3)$-approximation of the SMT of G_3 and has weight at most a constant times the weight of the SMT of G_3. Hence, G_4 is a planar banyan of the terminal set Z and so, our algorithm also delivers a planar banyan of a set of terminals among obstacles in the plane.

A note on the running time. Of course, the constants hidden in the O-notations above all depend on $1/\varepsilon$. Our algorithm builds the planar graph G_3 in time $\mathcal{O}(\frac{\kappa^4}{\varepsilon^{11}} n \log^2 n)$ and its size is more precisely $\mathcal{O}(\frac{\kappa^4}{\varepsilon^{11}} n \log n)$. The PTAS of Borradaile et al. takes time singly exponential in $1/\varepsilon$ [BKM09].

4.2 Correctness

We present two proofs for the correctness of Algorithm SMTO-PTAS. The first one results in better constants but does not work in the rectilinear case. The other one is more general and can even be partly extended to give us some structural information about SMTOs in higher dimensions but uses much larger constants. The proof technique and the generalization of the empty ball lemma used in the second proof might be interesting

in their own right. In this section, we present these two proofs for the Euclidean metric. In the next section, we discuss uniform orientation metrics where we include a simpler proof for the rectilinear case that results in small constants.

4.2.1 Key Differences

Compared to the obstacle-free case, we have two main new challenges: first, the fact that only some pairs of vertices are visible to each other; and second, that we do not know which corners will be included in the optimal Steiner tree, i.e. many vertices of the input need not be part of the SMTO at all. To deal with the former, we formulate and prove Lemma 4.8, which is a technical lemma that is very important in both of our proofs; it enables us to prove our generalizations of the hexagon property (Lemma 4.9) and the empty ball lemma (Lemma 4.15).

But more importantly, these issues make the so-called spanner-path property of Rao and Smith (Lemma 34 in [RS98]) invalid for our case. This property says that two vertices that are connected in the SMT by an edge of length L, can not be connected in a spanner by a chain of "tiny" edges of length $< L$. Indeed, in our case, two terminals and/or corners *can* be connected by a spanner-path consisting entirely of "tiny" edges, finding their way among obstacles. To overcome this problem, we introduce $\mathcal{O}(\log n)$ layers of grids around each edge of the spanner G_1: we know that any two vertices in the spanner are connected by a short path with at most n edges and one of them is long enough, so that by multiplying its length with a power of 2, we can produce a grid of candidate Steiner points around it that suits our purposes (Lemma 4.10 and Corollary 4.11). This technique does not require the spanner path property and also does not depend on finding a terminal or corner near a Steiner point A that is close to A *in the* SMTO or even part of the SMTO at all; it is sufficient to find a terminal or corner that is close and visible to A and this can be done using Lemmas 4.8 and 4.9.

4.2.2 First Proof

Our proof is structured as follows. First, we show through a number of lemmas (Lemmas 4.8-4.11) that near every "non-negligible" Steiner point, there exists a terminal, corner or grid-point that can approximate it well enough. Then we consider an optimal obstacle-avoiding Steiner tree of the input and show how to transform it into another obstacle-avoiding Steiner tree, that does not contain "negligible" Steiner points and that shares its vertices and edges with the graph G_3 constructed by SMTO-PTAS. We argue that this transformation can be done in such a way that the resulting Steiner tree has length at most $(1 + \varepsilon/2)$ times longer than the optimal Steiner tree. Hence, finding a $(1 + \varepsilon/3)$-approximate Steiner tree in G_3 gives us the desired result. In the following, we consider the notation of SMTO-PTAS and let $T^\star$ be an SMTO of the input.

We use the notation $d(A, B)$ for the length of the shortest obstacle-avoiding path between two points A and B and the notation $d_G(A, B)$ for the shortest path between A and B in a graph G. We denote the length of a graph G, i.e. the total sum of its edge-weights, by $\ell(G)$.

We start by mentioning some well known facts about Euclidean Steiner minimum trees. These facts were first proven for the obstacle-free case by Gilbert and Pollak [GP68] and later, stated and used for the case with obstacles by several authors [Pro88, ZW99] (explicit new proofs are not given but the generalizations are straightforward). In the following, recall that we use the term Steiner point for vertices of the tree that do not coincide with terminals or corners:

Fact 4.2. *The* SMTO *is a tree that includes all terminals as vertices. It might include corners or Steiner points as additional internal vertices.*

Fact 4.3. *A Steiner point of the* SMTO *may not occur on the boundary of some obstacle.*

Fact 4.4. *Every Steiner point of the* SMTO *has* 3 *incident edges making angles of* 120°.

Fact 4.5. *Every terminal and corner has degree at most* 3 *in the* SMTO.

Fact 4.6. *Two edges of the* SMTO *meet only at a common endpoint, i.e. the* SMTO *is not self-intersecting.*

Fact 4.7 (120° wedge property [Pro88])**.** *If s is a Steiner point of the* SMTO*, then in any closed* 120° *wedge with apex s, there exists a terminal or corner v and an* SMTO *path from s to v that lies entirely inside the wedge.*

The following lemma is of central importance for our work and is illustrated in Fig. 4.4:

Lemma 4.8. *Let S be a closed convex region of the plane and let $A \in S$ be a point that is not contained in the interior of any obstacle. Then, we have*
(i) a terminal or corner in S that is visible to A; or
(ii) the maximal visible area to A in S is a closed convex region $S' \subseteq S$ that contains no terminal or corner and its border is composed of parts of obstacle edges and parts of the boundary of S. Furthermore, any obstacle-avoiding path contained in S and connected to A is contained in S'.

Proof. Assume there exists an obstacle edge e passing through S that has both its endpoints outside of S. Let e^+ and e^- be the two closed half-planes induced by extending e to a straight line and w.l.o.g. assume $A \in e^+$. Then we know that e^- does not belong to

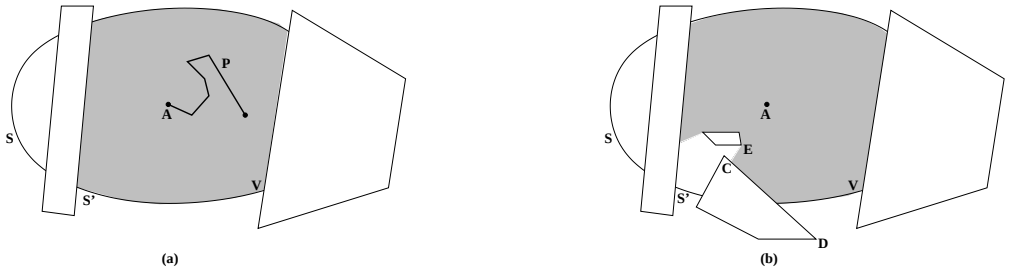


Figure 4.4: Illustration of Lemma 4.8; S is the large convex region partly hidden by the polygonal obstacles; the visible area to A in S, denoted by V, is shaded. (a) $S' = V$; (b) $S' \neq V$ and corner E is visible to A.

the visible area to A in S. Also, if P is an obstacle-avoiding path connected to A inside S, it can not cross e and thus, it is also completely contained in e^+. Define $S_1 = S \cap e^+$. S_1 is closed and convex, contains A, and we can repeat the argument above until we get a closed convex region $S' \subseteq S$ that contains the visible area to A in S and such that there are no obstacle edges passing "through" S' as mentioned above. Furthermore, S' shares its border with S except for finitely many straight-line segments where it may consist of obstacle edges. Also, any obstacle-avoiding path connected to A inside S is included in S' (Fig. 4.4(a)).

Let $V \subseteq S'$ be the area of S' visible to A. Suppose $V \neq S'$. Then there is an obstacle edge e on the border of V that is not part of the border of S'. Consider a line-segment l lying in V and connecting A to e. If me move one endpoint of l along e, at least in one direction we will not leave S' (otherwise e would be passing "through" S'). Moving in this direction, l either meets one corner of e or another obstacle edge e'; in the latter case, it must be that we also meet at least one corner of e'. In both cases, we have found a corner in S' that is visible to A (see Fig. 4.4(b)). Otherwise, we have $V = S'$ and thus, S' is either free of terminals and corners or contains a terminal or corner visible to A. □

We define an Smto edge AB to be *locally D-bounded* if when walking from A or B, and away from AB, for at most 3 Smto edges or until we encounter a terminal or corner (whichever comes first), all edges we pass have length at most D (see Fig. 4.5(a)). Now consider an Smto edge AB and some fixed distance D. Let H_A be the regular hexagon of side length D that has A as a corner, does not contain AB, and builds two 120° angles with AB, i.e. if we extend AB, it would cut H_A in half (see Fig. 4.5(b)). We have the following property:

Lemma 4.9 (Generalized hexagon property)**.** *Let AB be a locally D-bounded* Smto *edge. Then the regular hexagon H_A of side length D defined above contains a terminal or a corner that is visible to A (this terminal or corner could be A itself).*

Proof. Apply Lemma 4.8 with $S = H_A$. If we find a terminal or corner inside S visible to A we are done. So assume not. Let S' be the corner-free and terminal-free region asserted by statement (ii) of Lemma 4.8. Now consider Fig. 4.5(b). If any of the points s_1, t_1, s_2, t_2, s_3, or t_3 are terminals or corners, then the one closest to A (in terms of number of edges) is connected to A by an obstacle-avoiding path inside H_A and by Lemma 4.8, it is contained in S' and thus visible to A; a contradiction. So, all these points are Steiner points and by Fact 4.4, the situation is exactly as illustrated in Fig. 4.5(b) or symmetric to it. Now consider the 120° wedge placed at s_3 as shown. By Fact 4.7, this wedge contains a terminal or corner connected to s_3 by an Smto path and by Fact 4.6, this path must be contained in H_A. By Lemma 4.8, this path is even contained in S' and visible to A; again, a contradiction. □

Let AB be an Smto edge of length L. For given constants $c \geq 1$ and $\varepsilon_1 > 0$, we define AB to be *locally long* if it is locally cL/ε_1-bounded (see Fig. 4.5). Otherwise we call it *locally short*. The next lemma builds the heart of our proof of Theorem 4.1. It assures that near every locally long Smto edge AB, we find an edge, e, of the spanner G_1 that

(a) (b)

Figure 4.5: (a) If T is a terminal and all edges in this figure, possibly except for e, have length at most D, then e is locally D-bounded; if $D = cL/\varepsilon_1$, where L is the length of e, then e is locally long; (b) Proof of Lemma 4.9: the wedge at s_3 includes a path to V_A.

is long enough, so that a grid layer around e will enclose the points A and B; and short enough, so that the spacing of the grid layer does not introduce too large an error.

Lemma 4.10. *Let AB be a locally long* SMTO *edge of length L with some constants $c \geq 1$ and $0 < \varepsilon_1 \leq 1$ to be specified. Consider a run of* SMTO-PTAS *with a constant $\kappa \geq 8c + 2$. Then there exists an edge e of length ℓ in the $(1 + \varepsilon_1)$-spanner G_1 and an integer $0 \leq i \leq \lceil \log_2 n \rceil$, so that $L \leq 2^i \ell \leq \kappa L/\varepsilon_1$ and so that A and B are included in a circle of radius $\kappa 2^i \ell/\varepsilon_1$ around the midpoint of e.*

Proof. By the hexagon property above with $D = cL/\varepsilon_1$, we know that there exists a terminal or corner V_A inside H_A and a terminal or corner V_B inside H_B, so that V_A is visible to A and V_B is visible to B (note that V_A resp. V_B could be equal to A resp. B). Then we know that $L \leq d(V_A, V_B) \leq L + 4cL/\varepsilon_1 =: M$ (see Fig. 4.6(a)). Now consider the shortest path between V_A and V_B in the spanner G_1. It consists of at most n edges and its length is at least L and at most

$$\begin{aligned}(1 + \varepsilon_1)M &= L + 4cL/\varepsilon_1 + L\varepsilon_1 + 4cL \\ &= ((4c + 1)\varepsilon_1 + 4c + \varepsilon_1^2)L/\varepsilon_1 \leq \kappa L/\varepsilon_1\end{aligned}$$

if we choose $\kappa \geq 8c + 2$. Also, this path lies entirely inside a circle Q of radius $R := (1+\varepsilon_1/2)M$ around the midpoint of the edge AB, since a shortest path that starts inside H_A, leaves Q, and returns to H_B has length $> 2(R - M/2) = (1 + \varepsilon_1)M$ and so, would be too long to be needed for a $(1 + \varepsilon_1)$-spanner path. Hence, there exists an edge e of length ℓ on this path inside Q, so that $L/n \leq \ell \leq \kappa L/\varepsilon_1$ (see Fig. 4.6(a)). If $\ell < L$, one can choose an integer $0 < i \leq \lceil \log_2 n \rceil$ so that $L \leq 2^i \ell \leq 2L \leq \kappa L/\varepsilon_1$ otherwise choose $i = 0$. Also, since e is inside Q, the distance between the midpoint of e to A or B is at

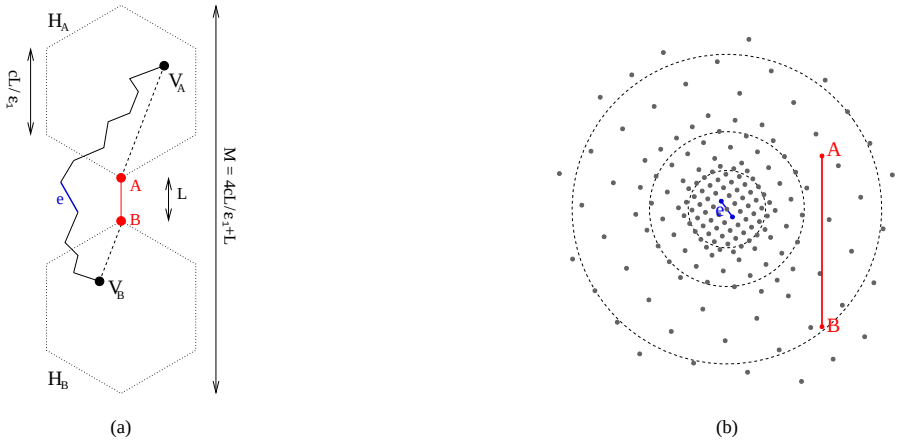


Figure 4.6: (a) Proof of Lemma 4.10: $L \le d(V_A, V_B) \le 4cL/\varepsilon_1 + L$ and one can find an edge e on the spanner path between V_A and V_B that is not too short and is close to A and B; (b) the candidate Steiner points around e "cover" the Steiner points A and B.

most

$$\begin{aligned} R + L/2 &= L + 4cL/\varepsilon_1 + L\varepsilon_1/2 + 2cL + L/2 \\ &= ((2c + 1.5)\varepsilon_1 + 4c + \varepsilon_1^2/2)L/\varepsilon_1 \le \kappa 2^i \ell/\varepsilon_1 \,. \end{aligned}$$ □

Let S_0 be the set of Steiner points of the SMTO $T^\star$ that are only incident to locally short edges of $T^\star$. By Fact 4.4 each vertex of S_0 is incident to exactly 3 locally short edges making angles of 120° with each other. Let S_1 be the set of all other Steiner points of $T^\star$. From Lemma 4.10 we get

Corollary 4.11. *Let A be a Steiner point from the set S_1 and let L be the length of a locally long edge incident to A. Then there exists a vertex A' of the graph G_3, so that $d(A, A') \le 2L\varepsilon_1$.*

Proof. Since A is incident to a locally long edge of length L of $T^\star$, we have by Lemma 4.10 that there exists an edge e of length ℓ in G_1 and a circle Q of radius $r = \kappa 2^i \ell/\varepsilon_1$ around the midpoint of e for some integer $0 \le i \le \lceil \log_2 n \rceil$, so that A is included in Q (see Fig. 4.6(b)). The grid of candidate Steiner points inside Q has spacing[3]

$$\delta = r\varepsilon_1^3/\kappa^2 = 2^i \ell \varepsilon_1^2/\kappa \le L\varepsilon_1$$

since $2^i \ell \le \kappa L/\varepsilon_1$ by Lemma 4.10. A technical lemma of Provan (Lemma 3.2 in [Pro88]) says that for a given Steiner point A in a δ-grid among obstacles, we can always find a terminal, corner or grid point A' that is visible to A, so that $d(A, A') \le 2\delta$. All terminals,

[3]The grid spacing in [RS98] is $r\varepsilon_1^2/\kappa^2$ but we believe that the exponent of ε_1 should be 3.

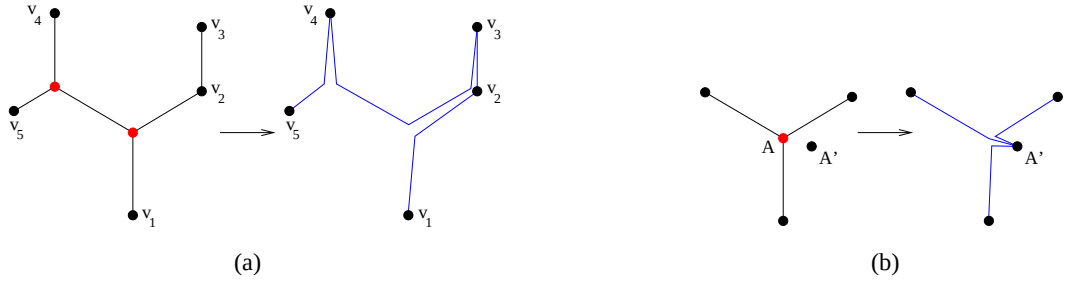


Figure 4.7: (a) Replacing a tree of locally short edges with an Euler tour; (b) moving a Steiner point A to its approximation A' in G_3.

corners, and candidate Steiner points are part of the graph G_3 and hence, our claim is proven. □

In Lemmas 4.12-4.14, we modify $T^\star$ into a tree $T_2^\star$ that is not much longer than $T^\star$ and whose vertex set is included in the vertices of the graph G_3. In the construction below, we do not require an edge to be a straight-line segment but allow edges that are paths comprised of a number of straight-line segments.

Lemma 4.12. *The total length of all locally short edges of $T^\star$ is at most $\varepsilon_1 \ell(T^\star)$.*

Proof. The total length of locally short edges can be estimated as follows: we charge the cost of a locally short edge of length ℓ, a *charger*, to its closest edge (in terms of number of edges) that has length $\geq c\ell/\varepsilon_1$. Now consider any edge of length L of $T^\star$. Any cost charged to it is caused by a charger that is at most 3 edges away, so by Facts 4.4 and 4.5 the number of these chargers is at most 28 and the amount charged by each of them is at most $L\varepsilon_1/c$. We set[4] $c = 28$ and so, the total sum is bounded by ε_1 times the length of $T^\star$. □

The main part of the proof of the next lemma is similar to the proof of the classic MST-based 2-approximation algorithm for the Smt problem [Cho78, Ple81]:

Lemma 4.13. *There exists an obstacle-avoiding Steiner tree $T_1^\star$ of the terminals such that*

1. *it contains all locally long edges of $T^\star$;*
2. *all of its Steiner points are from the set S_1;*
3. *the degree of each vertex is at most 4;*
4. *vertices of degree 4 are incident to at least 2 locally long edges of $T^\star$;*
5. $\ell(T_1^\star) \leq (1 + \varepsilon_1)\ell(T^\star)$.

Proof. Recall that the notion of locally long and locally short partitions the edges of $T^\star$ into two sets. Consider a maximal subtree T' of $T^\star$ that consists entirely of locally short

[4]Thus we can choose $\kappa = 8c + 2 = 226$.

edges and that contains at least one vertex of S_0. Let X_0 be the set of vertices of T' that belong to the set S_0 and let X_1 be the set of all other vertices of T'. Note that the vertices of X_0 are all Steiner points of degree 3 and by Fact 4.3, none of them lies on the boundary of an obstacle. Also, the vertices of X_1 all have degree 1 or 2 in T' (because they are adjacent to at least one locally long edge in $T^\star$). Consider an Euler tour C of T', starting at a leaf v_1 and turning right at each vertex (thus, turning around at each leaf), so that it visits each edge of T' exactly once in each direction. Let $v_1, \ldots, v_k$ be the vertices of X_1 in the order they are visited by C and let $C[i, j]$ denote the part of C that connects v_i and v_j. For each $i = 1, \ldots, k-1$, consider a path between v_i and v_{i+1} that closely follows $C[i, i+1]$ on its right side without touching it or any other part of $T^\star$; except for the special case when v_i and v_{i+1} are adjacent, in which case we may take the edge $v_i v_{i+1}$ instead (note that this can happen at most in one direction for each edge). Since all angles at the internal vertices of each $C[i, i+1]$ are less than 180°, such a path can be chosen so that it is not longer than $C[i, i+1]$ (see Fig. 4.7(a))[5]. Let T'' be the union of these $k-1$ paths and note that $\ell(T'') \leq 2\ell(T')$. Let $T_1^\star$ be the tree obtained from $T^\star$ by replacing each such maximal subtree of locally short edges by its corresponding set of paths as described above. This leaves all locally long edges of $T^\star$ untouched and removes all Steiner points of the set S_0 from $T^\star$. Also the degree requirements are satisfied, as one can easily check using Facts 4.4 and 4.5. The length of $T_1^\star$ is at most the length of $T^\star$ plus the total length of the locally short edges of $T^\star$. By Lemma 4.12, we get that $\ell(T_1^\star) \leq (1+\varepsilon_1)\ell(T^\star)$. □

Lemma 4.14. *There exists a Steiner tree $T_2^\star$ of the terminals, so that all its vertices are from the graph G_3 and we have $\ell(T_2^\star) \leq (1+13\varepsilon_1)\ell(T^\star)$.*

Proof. We start with the tree $T_1^\star$ from the Lemma above and modify it the following way. Each Steiner point A of $T_1^\star$ is incident to at least one locally long edge of $T^\star$. Let e be the shortest locally long edge incident to A and let L be its length. By Corollary 4.11, there exists a vertex A' of G_3 visible to A, so that $d(A, A') \leq 2L\varepsilon_1$. Moving the endpoints of each edge incident to A to A' adds at most $2L\varepsilon_1$ to the length of that edge (see Fig. 4.7(b)). Charge this amount to e. The degree of each Steiner point of $T_1^\star$ is either 3 or 4; so, each locally long edge is charged at most 4 times from each side. But if the degree of a vertex is 4, then there are at least 2 locally long edges incident to it, and we can charge each one of them twice (instead of charging one of them 4 times). So, each locally long edge of length L is charged at most 6 times and hence, the total overhead produced by moving the Steiner points to the vertices of G_3 is at most $12\varepsilon_1 \cdot \ell(T^\star)$. Thus, the length of $T_2^\star$ is at most $(1+13\varepsilon_1)\ell(T^\star)$. □

First proof of Theorem 4.1 for the Euclidean metric. By Lemma 4.14, we know that there exists a Steiner tree $T_2^\star$ with length at most $(1+13\varepsilon_1)\ell(T^\star)$ that uses only the vertices of G_3 as Steiner points. Since G_3 is a $(1+\varepsilon_2)$-spanner of its vertex set, we can replace each edge of $T_2^\star$ by a path in G_3 that is longer by a factor of at most $(1+\varepsilon_2)$. Note

[5] The reason we do not use shortest paths is that we do not want to change the degrees of other terminals or corners of the tree, nor introduce new Steiner points.

that this is also true if the considered edge is in fact a path comprised of a number of straight-line segments. This shows that the optimal Steiner tree $T_3^\star$ in the planar graph G_3 has length at most $(1+\varepsilon_2)(1+13\varepsilon_1)\ell(T^\star)$ and by choosing $\varepsilon_1 = \frac{\varepsilon}{52}$ and $\varepsilon_2 = \frac{\varepsilon}{6}$, we can ensure that this amount is at most $(1+\varepsilon/2)\ell(T^\star)$. The PTAS of Borradaile et al. [BKM09] returns a $(1+\varepsilon/3)$-approximate Steiner tree T_{out} of G_3 and we have

$$\ell(T_{\text{out}}) \leq (1+\varepsilon/3)\ell(T_3^\star) \leq (1+\varepsilon/3)(1+\varepsilon/2)\ell(T^\star) \leq (1+\varepsilon)\ell(T^\star)\,.$$

□

4.2.3 Second Proof

Our second proof for Theorem 4.1 is based on our generalization of the so called empty ball lemma. The proof of this lemma can be obtained by a straightforward generalization of the proofs found in [RS98, Lemma 36] and [Zha05, Lemma 5.4.5] (providing a very detailed proof):

Lemma 4.15 (Generalization of the empty ball lemma)**.** *Let S_1 and S_2 be closed convex regions in the plane whose interiors are free of terminals and obstacle edges but whose borders may partly consist of obstacle-edges. Denote the parts of their borders that are not obstacle-edges as the* free border*. Assume that S_2 encloses S_1 and that the distance between every point on the free border of S_1 to any point on the free border of S_2 is at least $\gamma > 0$. Then, for any obstacle-avoiding* Smt*, the number of Steiner points inside S_1 is bounded by a constant $s_0 \leq (96e)^8$ (where e is the base of the natural logarithm).*

Our second proof for Theorem 4.1 is similar in its basic idea to the first proof: for every locally long edge AB, we find terminals or corners V_A and V_B with $L/2 \leq d(V_A, V_B) \leq 4cL/\varepsilon_1 + L$ and apply the same argument as before to find vertices A' and B' in G_3 to approximate AB; afterwards, the rest of the argument follows.

But here, we have to change the definition of locally long as follows: an Smt edge AB is considered *locally long* if when walking from A or B for at most s_0 edges or until we reach a terminal or corner, all edges we encounter have length at most $\frac{cL}{s_0\varepsilon_1}$, where s_0 is the constant from Lemma 4.15. Hence, in order for our charging scheme to work, we have to choose $c = \mathcal{O}(2^{s_0})$. Here, we only provide the main different part of this proof; the other details can be straightforwardly adapted from the first proof.

Second proof of Theorem 4.1 for the Euclidean case, main part. Consider a locally long Smt edge AB of length L, let $R = cL/\varepsilon_1$ and consider three co-centric balls Q_1, Q_2, and Q_3 with radii $R + L/2$, $2R$, and $(1+\varepsilon_1/2)(4R+L)$ around the midpoint of AB, respectively. W.l.o.g. assume AB is vertical and A is above B. Let d_1 be a line perpendicular to AB, passing through A and let d_2 be parallel to d_1 passing at distance $L/4$ below A (see Fig. 4.8). Let S_1 be the part of Q_1 above d_1 and let S_2 be the part of Q_2 above d_2. We show that S_2 contains a terminal or corner V_A visible to A. Assume not. Then, by Lemma 4.8, the part of S_2 visible to A is a closed convex region S_2' free of terminals and corners. Similarly, let S_1' be the visible region of S_1 to A and note that $S_1' = S_2' \cap S_1$. By applying Lemma 4.15 to S_1' and S_2' with $\gamma = L/4$, we see that the

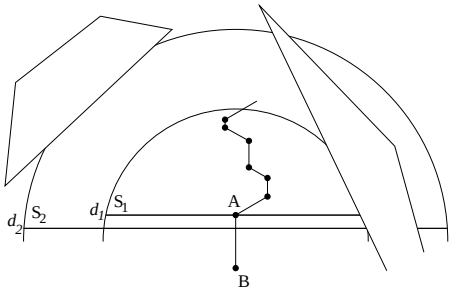


Figure 4.8: Illustration of the second proof. AB has length L and the distance between d_1 and d_2 is $L/4$. The inner circle has radius $R + L/2$ and the outer circle radius $2R$.

number of Steiner points inside S_1' is bounded by a constant s_0. We show that there exists an SMT path connected to A, leaving S_1' without crossing d_1: just notice that when we start walking at A and move from Steiner point to Steiner point, there is always an SMT edge that does not go "downwards", i.e. that builds an angle $90° \leq \alpha \leq 270°$ with AB. So, we can always find a next point until we leave S_1', so that we can encounter a terminal or corner. Obviously, this path has to leave S_1' at a part of its free border, so it has to leave Q_1, too, and thus, the length of this path is at least R. It consists of at most s_0 edges and hence, we can find an edge that has length at least R/s_0 and is at most s_0 edges away from A — a contradiction to the assumption that AB is locally long. So, the desired point V_A has to exist. Similarly one finds a point V_B visible to B and we will have $L/2 \leq d(V_A, V_B) \leq 4R + L$. The shortest path in the spanner G_1 connecting V_A and V_B will be completely included in Q_3 and the rest of the argument follows. □

The key advantage of this second proof is that it does not make use of the hexagon property and thus, can be easily extended to metrics that do not have a wedge-property, such as the rectilinear metric. Also, it gives us some structural information about SMTOs in higher dimensions; in fact, it shows that SMTOs in higher dimensions have similar properties and could potentially be handled with similar methods; but since the resulting graphs are typically not planar, our PTAS does not apply.

4.3 PTASes in Uniform Orientation Metrics

We first briefly discuss λ-geometries with $\lambda \geq 3$, where we will see that the proofs above readily generalize. Then we turn our attention to the rectilinear case that has different properties and requires other techniques. In the last part, we prove our generalization of Arikati et al.'s [ACC+96] planar spanner result to the cases with $\lambda \geq 3$.

4.3.1 The Cases $\lambda \geq 3$

Brazil et al. [BTW00] showed that for $\lambda \geq 3$, there always exists an SMT such that the minimum angle at a Steiner point is $90° \leq \alpha_{\min} \leq 120°$ and the maximum angle is $120° \leq \alpha_{\max} \leq 150°$. For these cases, we get an $\alpha_{\max}$-wedge property like Fact 4.7 of the Euclidean case and we can use it to prove an analog of the Hexagon property (Lemma 4.9). Also, using this $\alpha_{\max}$-wedge property one can derive the following generalization of Provan's lemma [Pro88]:

Lemma 4.16. *Let A be a Steiner point of an* SMT *with an α-wedge property for an $\alpha < 180°$ and assume there is a grid with spacing δ around A. Then there exists a grid point, terminal, or corner A' that is visible to A such that $d(A, A') \leq \delta / cos\frac{\alpha}{2}$.*

Using these two results, one can generalize both of our proofs from Section 4.2 straightforwardly to all λ-geometries with $\lambda \geq 3$, where again, the first proof results in much better constants (but possibly different ones from the Euclidean case). The only thing that remains is to show that the planar spanners needed for the algorithm exist and can be computed in the required time. This is shown in the last part of this section.

4.3.2 The Rectilinear Case

In the rectilinear case, we do not have an α-wedge property for an $\alpha < 180°$; in fact, 180°-angles can occur at any Steiner point. But instead, the structure of rectilinear Steiner trees is well-studied. Particularly, we have the following lemma, adapted from [Hwa76, PS02]:

Lemma 4.17. *Let Z be a set of terminals and O a set of disjoint rectilinear obstacles in the plane, so that in any obstacle-avoiding rectilinear Steiner minimum tree (*RSMTO*) of Z, all terminals are leaves and no corner is included except for corners that coincide with terminals. Let A and B be two Steiner points in an* RSMTO *of Z that are connected by a horizontal line-segment. Then B can not be connected to a third Steiner point by a vertical line segment.*

We define a *full component* of an RSMTO to be a subtree in which every terminal and corner is a leaf. We can decompose a given RSMTO into its full components and do the following as long as possible: replace every full component by another RSMTO that includes more corners or in which the terminals or corners have degree more than one. Then we will get an RSMTO $T^\star$ where we can apply Lemma 4.17 to each of its full components (counting the included corners as terminals) and so, $T^\star$ itself will have the property of the lemma (see Fig. 4.9(a)):

Corollary 4.18. *For any given set of terminals and disjoint rectilinear obstacles in the plane, there exists an* RSMTO*, so that for any two Steiner points A and B that are connected by a horizontal line-segment, B is not connected to a third Steiner point by a vertical line segment.*

Figure 4.9: (a) illustration of Lemma 4.17 and Corollary 4.18: if A and B are Steiner points, C must be a terminal or corner; (b) proof of Lemma 4.19: if the grid point A' is not visible to the Steiner point A, there must be an obstacle corner P closer and visible to A.

Now one can use Corollary 4.18 instead of the hexagon property to prove a rectilinear version of Lemma 4.10 with even better constants: namely, one can loosen the definition of locally long edges to include only edges that are at most 2 edges away and so, one can choose $c = 12$ and $\kappa = 4c + 2 = 50$. We also have

Lemma 4.19. *For a given set of terminals and disjoint rectilinear obstacles in the plane, there exists an obstacle-avoiding* RSMTO *that has the property of Corollary 4.18 and each of its Steiner points A fulfills: if there is a grid with spacing δ around A, then there exists a grid point, terminal, or corner A' that is visible to A, so that $d(A, A') \leq 2\delta$.*

Proof. The RSMTO from Corollary 4.18 can be chosen so that all Steiner points lie on the Hanan grid spanned by the terminals and corners [Han66, GC94]. Let A be a Steiner point on this grid. Then there is a terminal or corner visible to A both on the horizontal and the vertical line passing through A. Consider the two line segments connecting A to these terminals or corners. The grid point A' enclosed in the 90°-wedge built by these two line segments is either visible to A or is blocked by an obstacle that has a visible corner to A inside the rectangle spanned by A and A' (see Fig. 4.9(b)). □

With these observations, both of our proofs from the last section adapt straightforwardly to the rectilinear case with the constants mentioned above.

4.4 Planar Spanners for Uniform Orientation Metrics

Consider a λ-geometry and let $\omega = \pi/\lambda$ be the smallest allowed angle. We denote the metric induced by a λ-geometry by a *λ-metric*; the distance between two points in a λ-metric by the *λ-distance*; and a path that is constituted entirely of edges in legal directions of the given λ-geometry, a *λ-path*. Before we start with the construction of our spanner, we need the following technical lemma:

Lemma 4.20. *Consider a λ-geometry with smallest allowed angle ω and let a set of disjoint polygonal obstacles be given whose edges are parallel to the allowed directions. Consider two points A and B in the plane. Then there exists a shortest path (with respect to the given λ-metric) between A and B that passes through $A = v_0, v_1, v_2, \ldots, v_k = B$, so that each v_i with $0 < i < k$ is a corner and so that the path between each v_i and v_{i+1}*

Figure 4.10: Proof of Lemma 4.20: (a) Case (i); (b) Case (ii).

is either a straight line in an allowed direction or is comprised of two straight lines in two consecutive allowed directions, i.e. allowed directions that build an angle of $\pi - \omega$ *with each other.*

Proof. Let $d_1, \ldots, d_\lambda$ be the allowed directions building consecutive angles of ω. We consider a shortest path between A and B that uses the maximum possible number of obstacle-corners and among those, one that uses the least number of straight-line segments. If this path contains an obstacle-corner D (besides possibly A and B), we can apply our proof to the subpaths AD and DB independently and we are done. So assume that our selected path does not contain any corners besides possibly A and B. We distinguish three cases:

Case (i): Assume A and B are visible to each other. If AB is a legal direction, we are done. Otherwise, w.l.o.g. assume AB lies between directions d_1 and d_2. Let $\triangle ACB$ be the triangle in which AC is parallel to d_1 and CB is parallel to d_2. We call this triangle the λ-*triangle* of AB. Clearly, the path ACB is a shortest path connecting A and B. We first show by induction on the number of corners inside $\triangle ACB$, that A and B can be connected by a shortest path using only directions d_1 and d_2 inside $\triangle ACB$. If $\triangle ACB$ contains no parts of an obstacle, we are done. Otherwise, since obstacle-edges are assumed to be only in legal directions and since d_1 and d_2 are consecutive directions and AB is not crossed by any obstacle-edge, the triangle must include at least one corner. Sweep a line parallel to d_2 *along* AC until we hit an obstacle-corner (if we hit an edge, we will also hit a corner by the preceding observation). Let D be the point where the swept line leaves AC and let E be the point where it meets AB (see Fig. 4.10(a)). Then ADE is a shortest path connecting A and E and using only directions d_1 and d_2. Furthermore, E and B are visible to each other and the λ-triangle of EB is contained in $\triangle ACB$ and contains at least one corner less. So, we get our claim by the induction hypothesis. Now we know that either A and B can be connected by the two straight-line segments AC and CB or there exists a shortest path between A and B including another corner — a contradiction to the selection of our path.

Case (ii): Assume A and B are not visible to each other but are connected by exactly 2 straight-line segments AC and CB (in legal directions, of course). Since the line AB is crossed by an obstacle-edge, but no such edges cross AC and CB, the triangle $\triangle ACB$ must include at least one obstacle-corner. Circle a line l *around* A starting at AC and

moving towards AB until it hits an obstacle corner D inside $\triangle ACB$. Consider the parallelogram between the points C and D with lines parallel to AC and CB. This parallelogram is completely "swept over" by l and hence, contains no obstacles. Let P_1 and P_2 be the other two corners of this parallelogram. Replacing P_1C and CP_2 with P_1D and DP_2 results in a shortest path between A and B that includes an additional corner — again, a contradiction (see Fig. 4.10(b)).

Case (iii): Assume A and B are not visible to each other and the path between them is comprised of more than 2 straight-line segments. By an argumentation similar to that in Case (ii), one can show that the first 3 line segments can either be replaced by a path that contains an additional obstacle-corner or they can be replaced by at most 2 line segments — in both cases, a contradiction to the selection of our path. □

Note that Lemma 4.20 implies that if we consider the visibility graph of a given set of terminals and corners and measure the length of each edge in the given λ-metric, then the shortest path between any two terminals or corners is included in the visibility graph. Furthermore, given the shortest path between a pair of points in the visibility graph, one can construct an obstacle-avoiding λ-path of the same length between those points. Using these observations, one can easily adapt Clarkson's spanner algorithm [Cla87] to work for all λ-geometries.

But Clarkson's spanner is not necessarily planar. Arikati et al. [ACC$^+$96, Zeh02] show how to find a planar rectilinear $(1+\varepsilon)$-spanner of the visibility graph among disjoint polygonal obstacles in the plane that uses at most $\mathcal{O}(n)$ Steiner points in time $\mathcal{O}(n \log n)$. This spanner might include obstacle-edges that are not rectilinear but their length is measured in the rectilinear metric. We first show that one can rotate the axes of the coordinate system to build an arbitrary angle and still obtain such a spanner. Let d_1 and d_2 be two distinct directions in the plane, building an angle of ω with each other. Define a *d_1d_2-geometry* to be the geometry where one is allowed to move only in directions d_1 and d_2 and call the induced metric a d_1d_2-metric. We have the following lemma:

Lemma 4.21. *Let a set of terminals and a set of disjoint polygonal obstacles with a total of n vertices be given. For distinct directions d_1 and d_2, one can find a planar $(1+\varepsilon)$-spanner of the visibility graph with respect to the induced d_1d_2-metric in time $\mathcal{O}(n \log n)$ and of size $\mathcal{O}(n)$.*

Proof. Let $\{e_1, e_2\}$ be the standard orthonormal basis of the plane and assume d_1 and d_2 are unit vectors in the given directions. Let T be the linear transformation in the plane that maps d_1 to e_1 and d_2 to e_2. For any given vector $v = v_1e_1 + v_2e_2 = v_1'd_1 + v_2'd_2$, we know on one hand, that the length of v in the d_1d_2-metric is $|v_1'| + |v_2'|$ and on the other hand we have $Tv = v_1'e_1 + v_2'e_2$, i.e. we obtain the d_1d_2-length of v as the rectilinear length of Tv. Thus, we may apply T to the plane, find a rectilinear spanner of the visibility graph using the algorithm of Arikati et al. [ACC$^+$96], and apply T^{-1} to the result to obtain a d_1d_2-spanner of the visibility graph (see Fig. 4.11). □

We need one more observation before we can proceed with the main construction. The simple lemma below can easily be shown using the law of sines and the fact that

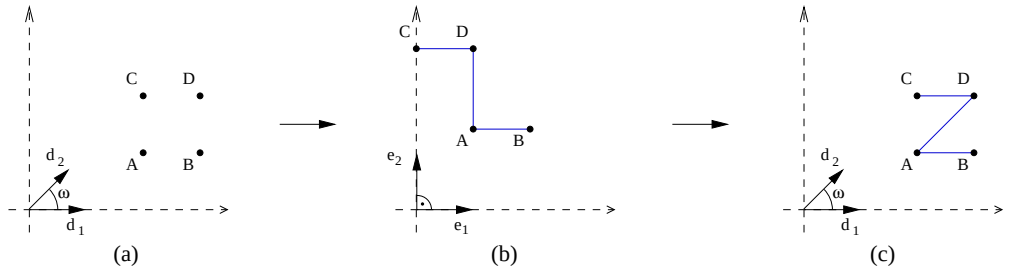


Figure 4.11: Proof of Lemma 4.21: we transform the input given in (a) and obtain (b); then we find a rectilinear spanner in (b) and transform it back to obtain the $d_1 d_2$-spanner shown in (c).

for $0 \leq \alpha \leq \pi/3$, we have $\sin \alpha \geq \frac{\alpha}{2}$:

Lemma 4.22. *For $\lambda \geq 3$, let d_1 and d_2 be vectors building an angle of π/λ with each other. Let d_1' and d_2' be two other vectors with the same property (not necessarily distinct from d_1 and d_2). If P is a parallelogram with sides parallel to d_1 and d_2 and maximum side length r, then there exists a parallelogram P' with sides parallel to d_1' and d_2', so that P' encloses P and each of its sides has length at most $\frac{4\lambda}{\pi} \cdot r$.* □

Now we can use a similar trick to the one used by Arikati et al. to obtain their Euclidean spanner: let $d_1, d_2, \ldots, d_\lambda$ be the allowed directions, so that two consecutive ones build an angle of π/λ with each other. For simplicity, define $d_{\lambda+1} = d_1$. Find $(1+\varepsilon)$-spanners $G_1, \ldots, G_\lambda$, so that G_i uses only edges parallel to d_i and d_{i+1} using Lemma 4.21. Let G be the graph obtained by superimposing all these spanners on each other, i.e. putting them on each other and adding all intersection points as new vertices to the graph. Now one can adapt the proof of Arikati et al. for the Euclidean case — published in the thesis of Zeh [Zeh02] — to show that G will still have $\mathcal{O}(n)$ vertices and is computable in $\mathcal{O}(n \log n)$-time. This is done by utilizing the special structure of the spanners. Each spanner is based on a division of the plane into $\mathcal{O}(n)$ regions, each containing a grid of constant size; using Lemma 4.22 and a similar argumentation as in [Zeh02], one can show that for two of these spanners, the number of regions R' of one spanner that overlap with a fixed region R of the other spanner, so that the maximum side-length of R' is longer than the maximum side length of R, is bounded by a constant. Hence, the total number of new vertices is linear in n and since G is planar, the same is true for the number of edges (assuming λ as a constant, of course).

Also, by Lemma 4.20, G is indeed a $(1+\varepsilon)$-spanner of the visibility graph: an approximate shortest path between each v_i and v_{i+1} of the lemma lies entirely in a spanner G_j. Thus, we have

Theorem 4.23. *Consider a λ-geometry and let a set of terminals and a set of disjoint polygonal obstacles whose edges are in the allowed directions be given, so that the total number of terminals and corners is n. Then one can find a planar $(1+\varepsilon)$-spanner (with*

respect to the metric of the λ-geometry) of the visibility graph of size $\mathcal{O}(n)$ in $\mathcal{O}(n \log n)$ time that uses only edges in the allowed directions.

4.5 Conclusion and Outlook

We have shown how recent progress in graph algorithms provides new insights and methods in geometric algorithms by presenting a near linear time approximation scheme for a geometric version of the Steiner tree problem — namely, the SMT problem among obstacles in the plane — using the recent PTAS of Borradaile et al. [BKM09] for the Steiner tree problem in planar graphs. To this end, we applied and modified recent techniques in the design and analysis of algorithms, most notably from the work of Rao and Smith [RS98] and that of Arikati et al. [ACC+96], and introduced new ones, see Sections 4.1-4.3, to obtain a planar graph of small size that still contains a sufficiently good approximation of the desired solution.

One interesting open question is whether it is possible to further reduce the time complexity of this problem to obtain an $\mathcal{O}(n \log n)$-approximation scheme. Note that even if the PTAS for planar graphs can be modified to run in linear time, we still need $\mathcal{O}(n \log^2 n)$ time to construct our planar spanner. One possible way to attack this problem is trying to intertwine the steps of the PTAS with the steps of our algorithm. Indeed, with some modifications one can achieve a running time of $\mathcal{O}(t \log t + k \log^2 k)$, where t is the number of terminals and k is the number of obstacle corners. But achieving a running time of $\mathcal{O}(n \log n)$ seems more difficult and is left as a subject for future work.

Part II

Algorithms on Minor-Closed Graph Classes

5 A Linear-Time Shortest-Paths Algorithm for H-Minor-Free Graphs[1]

The single-source shortest-paths problem with nonnegative edge-weights is one of the most-studied problems in computer science, because of both its theoretical and practical importance. Dijkstra's classical algorithm [Dij59] has ever since its discovery been one of the best choices in practice. Also from a theoretical point of view, until very recently, it had the best running time in the addition-comparison model of computation, namely $\mathcal{O}(m + n \log n)$ using Fibonacci heaps [FT87]. Pettie and Ramachandran [PR05] improved the theoretical running time in undirected graphs for the case when the ratio r between the largest and smallest edge-weight is not too large. They achieve a running time of $\mathcal{O}(m\alpha(m, n) + \min\{n \log n, n \log \log r\})$, where $\alpha(m, n)$ is the very slowly growing inverse-Ackermann function. Goldberg [Gol01] proposed an algorithm that runs on average in linear time. For the case of integer edge-weights, Thorup [Tho99] presented a linear-time algorithm in the word RAM model of computation, where the bit-manipulation of words in the processor is allowed. Hagerup [Hag00] extended and simplified Thorup's ideas to work for directed graphs in nearly linear time. But the question whether the standard addition-comparison model allows shortest-paths computation in worst-case linear-time is still open. For a fairly recent survey about shortest-paths algorithms, see Zwick [Zwi01].

For planar graphs, Henzinger et al. [HKRS97] presented the first linear-time algorithm to calculate shortest-paths with nonnegative edge-weights. Their algorithm works on directed graphs. It is based on Frederickson's [Fre85, Fre87] work who gave an $\mathcal{O}(n\sqrt{\log n})$-time algorithm for this case and whose idea was in turn based on Lipton and Tarjan's planar separator [LT79] to decompose the graph. Henzinger et al. first decompose the graph into a recursive division and then use this division to relax the edges in a certain order that guarantees linear running time. They claim that their algorithm can be adapted to work for any proper minor-closed family of graphs where small balanced separators can be found in linear time. Recently, Reed and Wood [RW09] improved the quadratic-time separator of Alon et al. [AST90] and showed that all proper minor-closed graph classes can be separated in linear time; so, we should be done. *However, both Frederickson's algorithm and Henzinger et al.'s algorithm assume that the graph has maximum degree* 3*; while this property can be achieved easily for planar graphs, we argue that it can not be achieved by standard methods for arbitrary minor-closed classes* (in particular, it can not be applied to apex graphs, i.e. planar graphs augmented by a "super-source"; these graphs have frequent application in the literature). We show how to build an appropriate recursive division of a graph from a proper minor-closed family

[1] This chapter is based on joint work with Matthias Müller-Hannemann [TM08, TM09b].

in linear time by a non-trivial extension of the algorithm in [HKRS97]. Our algorithm works for graphs with arbitrary degrees. But even after having the recursive division, the shortest paths algorithm in [HKRS97] depends on the assumption that the graph has bounded degree (and contains only a single source labeled initially with distance zero, cf. apex graphs). Using our recursive division, we show how to transform the graph and its division to have maximum degree 3, so that Henzinger et al.'s shortest-paths algorithm can be applied. Our modifications lead to the first shortest-paths algorithm for all proper minor-closed classes of graphs that runs in linear time in the addition-comparison model of computation.

As an application, we show how to implement Mehlhorn's 2-approximation algorithm for the Steiner tree problem [Meh88] (see Section 1.2) in linear time on proper minor-closed graph classes. No better time bound than Mehlhorn's own implementation of $\mathcal{O}(m + n \log n)$ has previously been known even for planar graphs. An important observation that we made is that Mehlhorn's distance network is a minor of the given graph, and thus its minimum spanning tree can be calculated in linear time with the algorithm of Mares [Mar04] (or that of Cheriton and Tarjan [CT76] in the planar case).

Contribution and Outline of this Chapter

The area of graph minor theory has been constantly evolving in the past three decades, especially due to the work of Robertson and Seymour [RS04] in the 1980's. Many important algorithms and meta-algorithms have been presented for large problem families on minor-closed graph classes, see e.g. [Gro03, DFHT05, DH05a, Gro07a]; and numerous theoretical concepts have been developed to handle them, e.g. the whole graph minor series and, say, [DGJT99, RW08, RW09]. We present the first linear-time algorithms for two fundamental graph-theoretic problems in these classes.

Our contribution can be summarized as follows:

- *identifying* that there is a gap in generalizing Henzinger et al.'s recursive division and shortest paths algorithms to all proper minor-closed graph classes;
- arguing that the gap *can not* be closed by standard methods;
- *filling in* the gap using deep results in graph minor theory;
- (re)proving in detail the *correctness* of the modified algorithm;
- showing how to modify a graph and its recursive division to obtain a bounded-degree graph and a recursive division with the same properties, resulting in the *first linear-time shortest-paths algorithm* for proper minor-closed graph classes;
- obtaining a useful application, namely, the *first linear time Steiner tree approximation*, which was previously not even known for planar graphs.

In Section 5.1, we review some needed concepts and previous work; in Section 5.2, we present our main result about shortest paths and in Section 5.3, the application to Steiner tree approximation.

5.1 Concepts and Algorithms from Previous Work

In this section, we review some concepts and some previous results that are needed in this chapter. These include graph minors, vertex partitioning, graph decomposition, and Henzinger et al.'s [HKRS97] single-source shortest-paths algorithm. Unless otherwise mentioned, n denotes the number of vertices and m the number of edges of a graph G.

5.1.1 Graph Minors

We consider classes of H-minor-free graphs, i.e. graphs that exclude a fixed graph H as a minor, as introduced in Section A.2. These are exactly the proper minor-closed classes of graphs, such as planar graphs, bounded-genus graphs, linklessly embeddable graphs, knotlessly embeddable graphs, and apex graphs. Note that for a proper minor-closed class of graphs, we can always consider the number of vertices ℓ of the smallest excluded minor and conclude that the complete graph $\mathbf{K}_\ell$ is a particular excluded minor of the class. Thus, the class of $\mathbf{K}_\ell$-minor-free graphs includes the considered minor-closed class of graphs. In the rest of this work, we work with $\mathbf{K}_\ell$-minor-free graphs, where ℓ is a fixed constant.

It follows from a theorem of Mader [Mad67] that $\mathbf{K}_\ell$-minor-free graphs have constant average degree, for some constant depending on ℓ. This, in turn, implies that these classes of graphs are sparse, i.e. we have $m = \mathcal{O}(n)$.

5.1.2 Vertex Partitioning

In [Fre85], Frederickson presented a simple algorithm called FindClusters, based on depth-first search, that given an undirected graph G *of maximum degree* 3 and a parameter $z \leq n$, partitions its vertex set into *connected components* each having at least z and at most $3z$ vertices. Since the algorithm gives us connected components, we can contract each one of them and get a minor of the input graph with at most n/z vertices. Frederickson used this algorithm to derive fast algorithms for the minimum spanning tree and shortest-paths [Fre87] problems. If a weighted graph does not have maximum degree 3, one can apply the following transformation: replace a vertex v of degree $\deg(v)$ with a zero-weight path of length $\deg(v)$ such that each edge incident to v is now incident to exactly one vertex of the path; i.e. we can *split* v using zero-weight edges. A similar transformation can be applied to directed graphs, too, using an additional zero-weight edge to complete a directed cycle. If the given graph is embedded in a surface, one can order the edges around the path/cycle in the same way they were ordered around the corresponding vertex in the given embedding. This way, the transformed graph will also be embedded in the same surface. However, for an arbitrary minor-closed class of graphs (e.g. apex graphs), it might not always be possible to remain in the class after transforming the graph this way, see Section 5.2. But Frederickson's FindClusters *depends* on the graph having bounded degree. Any constant bound would suffice for our purposes but in general such a bound does not exist for arbitrary H-minor-free graphs.

Reed and Wood [RW09] introduced an alternative partitioning concept that can be

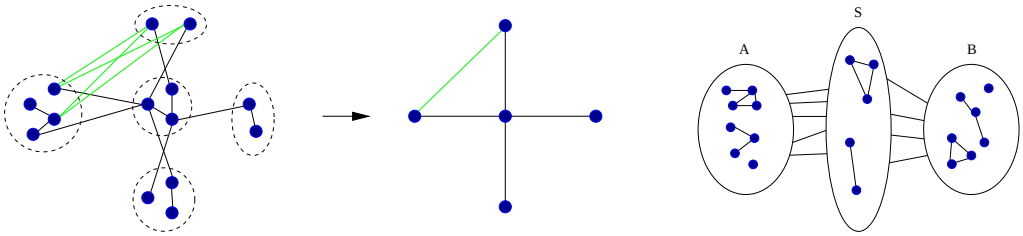


Figure 5.1: (a) Example of a connected H-partition and the result of collapsing its parts; each set of vertices enclosed by a dotted line is one part of the H-partition; (b) a balanced separator (A, B, S).

applied to a graph $G = (V, E)$ with arbitrary degrees excluding a fixed minor. Consider some partitioning $\mathcal{P} = \{P_1, \ldots, P_t\}$ of the vertex set V. Let $H = (V_H, E_H)$ be the graph obtained by *collapsing* every part P_i of G into a single vertex $v_i \in V_H$ $(1 \leq i \leq t)$ and removing loops and parallel edges. This way, there is an edge between two vertices v_i and v_j of H if and only if there is an edge between a vertex of P_i and a vertex of P_j in G $(1 \leq i < j \leq t)$. We say $\mathcal{P}$ is a *connected H-partition* of G if $v_i v_j \in E_H$ if and only if there is an edge of G between every connected component of P_i and every connected component of P_j (see Figure 5.1 (a)). Reed and Wood prove:[2]

Lemma 5.1 ([RW09])**.** *There is a linear-time algorithm that given a constant z and a graph G excluding a fixed $\mathbf{K}_\ell$-minor, outputs a connected H-partition $\mathcal{P} = \{P_1, \ldots, P_t\}$ of G such that $t \leq n/z$, and $|P_i| < c_0 \cdot z$ for all $1 \leq i \leq t$, where c_0 is a constant depending only on ℓ.*

Note that by contracting each connected component of each P_i in G to a single vertex, one gets a graph that contains an isomorphic copy of H as a subgraph; so, H is a minor of G and in particular, is also $\mathbf{K}_\ell$-minor-free. Hence, when dealing with graphs with unbounded degree, Lemma 5.1 can be used instead of FindClusters to partition the graph and reduce its size while keeping it free of some fixed minor.

Corollary 5.2. *Let G be a graph with n vertices excluding a fixed $\mathbf{K}_\ell$-minor, and let $c_0 = 2^{\ell^2+1}$ be a fixed constant. There exists a linear-time algorithm* H-Partition(G, z, ℓ) *with the following properties:*

- *it partitions the vertices of G into at most n/z sets;*
- *each set has at most $c_0 z$ vertices;*
- *it collapses each set into a single vertex, creating a new graph G';*
- *G' is a minor of G with at most n/z vertices.*

[2] We substitute $c_0 := 2^{\ell^2+1}$ and $z := 2k/c_0$ in [RW09, Lemma 4.1].

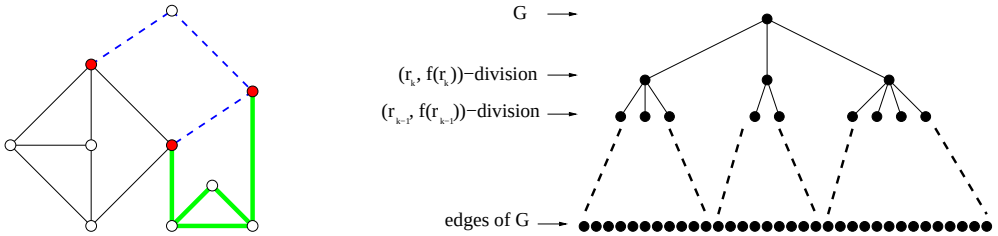


Figure 5.2: (a) An (r, s)-division with 3 regions indicated with different line styles and colors; the boundary vertices are filled with a red color; (b) a recursive division tree.

5.1.3 Graph Decomposition

A *separator* of a graph G is a partition of its vertices into three classes (A, B, S) so that there are no edges between A and B. The *size* of a separator is the size of the set S. We say that a separator is γ-*balanced* if for every connected component C of $G - S$, we have $|C| \leq \gamma|V|$. In this chapter, we use the term *balanced separator* to refer to a $\frac{1}{2}$-balanced separator of a graph (see Figure 5.1 (b)). Note that given a balanced separator (A, B, S), we can rearrange the connected components of $G-S$ so as to obtain a balanced separator (A', B', S) with $|A'|, |B'| \leq \frac{2}{3}|V|$; hence, we may assume this property without loss of generality. For a function $f : \mathbb{N} \to \mathbb{N}$, a subgraph-closed class of graphs is said to be f-*separable* if every n-vertex graph in the class has an $\mathcal{O}(f(n))$-size balanced separator. Reed and Wood [RW09] showed that all $\mathbf{K}_\ell$-minor-free graphs are f-separable in *linear* time for $f(n) = \mathcal{O}(n^{2/3})$. For planar graphs, one can use the seminal planar separator theorem of Lipton and Tarjan [LT79], obtaining an $\mathcal{O}(\sqrt{n})$-size balanced separator in linear time.

An (r, s)-*division* of an n-vertex graph is a partition of the *edges* of the graph into $\mathcal{O}(n/r)$ sets, called *regions*, each containing $r^{\mathcal{O}(1)}$ vertices and each having at most s *boundary vertices* (i.e. vertices that occur in more than one region; see Figure 5.2 (a)). For a nondecreasing positive integer function f and a positive integer sequence $\overline{r} = (r_0, r_1, \ldots, r_k)$, an $(\overline{r}, f)$-*recursive division* of an n-vertex graph is defined as follows: it contains one region R_G consisting of all of G. If G has more than one edge and $\overline{r}$ is not empty, then the recursive division also contains an $(r_k, f(r_k))$-division of G and an $(\overline{r}', f)$-recursive division of each of its regions, where $\overline{r}' = (r_0, r_1, \ldots, r_{k-1})$. A recursive division can be represented compactly by a *recursive division tree*, a rooted tree whose root represents the whole graph and whose leaves represent the edges of the graph (see Figure 5.2 (b)). Every internal node represents a region, namely, the region induced by all the leaves in its subtree. The children of a node of the tree are its immediate subregions in the recursive division.

Using Frederickson's partitioning [Fre85] and division [Fre87] methods, Henzinger et al. [HKRS97] present a linear-time algorithm to find certain recursive divisions in planar graphs: they determine a vector $\overline{r}$ and an $(\overline{r}, cf)$-recursive division of the graph for some

constant c, such that the inequality

$$\frac{r_i}{f(r_i)} \geq 8^i f(r_{i-1}) \log r_{i+1} (\sum_{j=1}^{i+1} \log r_j) \tag{5.1}$$

is satisfied for all r_i's exceeding a constant. The obtained recursive division tree has $\mathcal{O}(n)$ nodes and its depth is roughly $\mathcal{O}(\log^\star n)$. The idea of the algorithm is as follows: first, iteratively reduce the size of the graph by partitioning the vertices of the graph (using Frederickson's FindClusters) and building minors; then, working backwards, find (r, s)-divisions of the smaller graphs (for appropriate values of r and s), imposing divisions on the larger graphs and at the same time building the recursive division tree. Since the time-consuming calculation of (r, s)-divisions is done on the smaller graphs, they succeed to prove that the overall time complexity is linear.

5.1.4 Single-Source Shortest-Paths on Planar Graphs

Henzinger et al. prove the following theorem:

Theorem 5.3 ([HKRS97])**.** *Let a graph G be given with maximum in-/outdegree 2 and assume that G is equipped with an $(\overline{r}, cf)$-recursive division tree, for some constant c, so that inequality (5.1) is satisfied for all r_i's exceeding a constant. Then, the single-source shortest-paths problem with nonnegative edge-weights can be solved on G in linear time.*

To prove this theorem, they use a complicated charging scheme that also depends on the graph having *a single source and bounded degree*. Together with the result from the previous subsection, it follows that single-source shortest-paths with nonnegative edge-weights can be calculated in linear-time on planar graphs.

5.2 Single-Source Shortest Paths on H-Minor-Free Graphs

In this section, we prove our main theorem about shortest paths:

Theorem 5.4. *In every proper minor-closed class of graphs, single-source shortest-paths with nonnegative edge-weights can be calculated in linear time.*

First, we argue that the degree requirement of Henzinger et al.'s algorithm can not be fulfilled by standard methods for arbitrary minor-closed classes of graphs. By "standard methods" we mean the following: for a given vertex v in a graph and a permutation π_v of its neighbors (which we think of as an ordering of the neighbors), let the operation $\textsc{Split}_0(v, \pi_v)$ be defined as replacing v by a path P_v of length $\deg(v)$ of zero-weight edges and connecting each of the neighbors of v to exactly one vertex of P_v in the order specified by π_v. This way, every vertex on P_v will have degree exactly 3, and hence this operation can be applied to every vertex of a given graph to establish a maximum degree of 3 (while preserving shortest paths).

In Subsection 5.1.2 we discussed a particular choice of the permutation π_v for embedded graphs, namely, using the (cyclic) order of the neighbors of a vertex as given by the

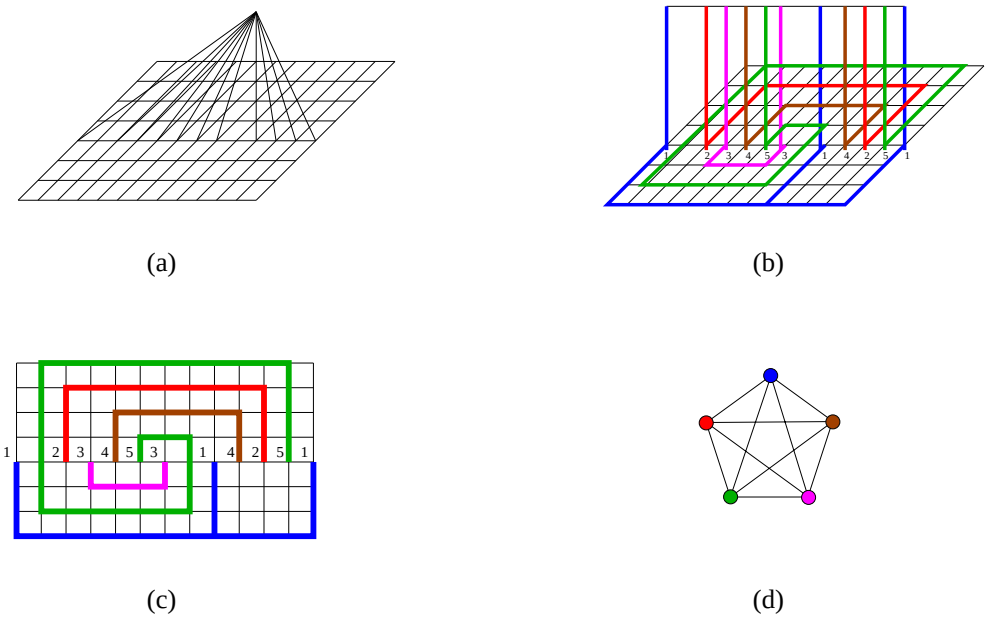


Figure 5.3: A simplified example for the proof of Proposition 5.5: the apex of the graph in (a) is split, resulting in the graph (b); the vertices are labeled and connected according to the disjoint trees in (c); contracting the thick edges in (b) results in a $\mathbf{K}_5$-minor (d).

embedding; this way, we could ensure that after the operation $\textsc{Split}_0$ is applied, the resulting graph is still embedded on the same surface. In this section, we show:

Proposition 5.5. *For every given $k \in \mathbb{N}$, there exists a $\mathbf{K}_6$-minor-free graph G_k that contains a vertex v such that* for any permutation π_v*, the graph resulting from applying* $\textsc{Split}_0(v, \pi_v)$ *contains $\mathbf{K}_k$ as a minor.*

The key lies in the observation that splitting an apex might introduce arbitrarily large minors. This is a well-known fact in graph minor theory [RS03]. For the sake of completeness, we include a short proof below. Apices are a fundamental part of minor-closed graph classes as is demonstrated by the powerful graph-decomposition theorem of Robertson and Seymour [RS03]. This theorem shows, in a sense, that at most a bounded number of apices are allowed in these classes; and intuitively, splitting an apex with unbounded degree might result in an unbounded number of apices and is thus not allowed in general.

Proof Sketch of Proposition 5.5. We define G_k to be a sufficiently large planar grid-graph augmented by an apex as follows: consider a sequence S of numbers between 1 and k such that each possible pair of these k numbers is at least once adjacent in S. Let $t < k^2$ be the length of this sequence. Choose a set W of t vertices in the grid that are sufficiently far away from each other and add an apex v connected to these t vertices. This completes the definition of G_k, which is clearly $\mathbf{K}_6$-minor-free.

Now by applying $\textsc{Split}_0(v, \pi_v)$, for any given permutation π_v, the apex v becomes a path P_v of t vertices, each one connected to exactly one vertex of W. This path imposes an order on the vertices in W. We label the vertices in W according to this order using the sequence S. Let W_i be the set of vertices in W labeled by i ($1 \leq i \leq k$). For each i, construct a tree T_i that connects the vertices of W_i in the planar grid. Note that if the grid is sufficiently large and the vertices in W are sufficiently far away from each other, it is easily possible to choose the trees T_i to be all disjoint. Let U be the set of edges connecting the vertices in W with P_v. Now, if we contract the trees T_i and the edges in U and delete redundant edges, what remains is a $\mathbf{K}_k$-minor (see Figure 5.3). □

5.2.1 Our Generalized Recursive Division Algorithm

Our modified algorithm is given in Algorithm 5.2.1. Our modifications are only in three places but as we already discussed, they are essential to make the algorithm work for all proper minor-closed graph classes. In this subsection, we discuss the algorithm and our modifications thereof in detail and in the next subsection, we present its proof of correctness.

Let the input graph be $G = (V, E)$. In the first phase of the algorithm, the input graph is reduced in size by building minors. Specifically, starting with $G_0 = G$, a sequence of graphs $G_0, G_1, \ldots, G_{I+1}$ is built, so that for $i > j$, G_i is a minor of G_j and G_{I+1} is the first graph in the sequence having less than $n/\log n$ vertices. To this end, a sequence of parameters z_i is used to specify the size of the next graph in the sequence as follows: let n_i be the number of vertices of G_i. In the original algorithm, the sequence is defined as $z_0 = 2$ and $z_{i+1} = 7^{z_i^{1/5}}$ and has the effect that G_i is partitioned into at most n_i/z_i connected components, each one having at most $3z_i$ vertices (using Frederickson's FindClusters [Fre85]). Each of these components is contracted to construct G_{i+1}, a minor of G_i with $n_{i+1} \leq n_i/z_i$ vertices.

Our first two changes occur in this phase of the algorithm. First, instead of using Frederickson's FindClusters, we make use of the H-Partition procedure of Reed and Wood [RW09] to achieve a similar effect without depending on the graph having bounded degree (cf. Section 5.1.2). Secondly, we had to change the definition of the z_i's to be $z_0 = 2$ and $z_{i+1} = 14^{z_i^{1/7}}$. This is due to the fact that in order to prove inequality (5.1) in our case, we need the exponent of z_i to be $\frac{1}{7}$ instead of $\frac{1}{5}$; but then, in order to ensure that the z_i's still grow (extremely fast) towards infinity, the base of the exponentiation had to be changed from 7 to 14, too. Indeed, 14 is the smallest integer that can be used, so that the z_i's grow towards infinity. Now, using the H-Partition procedure as in Corollary 5.2, we still have that $n_{i+1} \leq n_i/z_i$ but now, each vertex of G_{i+1} represents at most $c_0 z_i$ vertices of G_i (instead of the original $3z_i$).

In the second phase, the algorithm works backwards from G_{I+1} towards G_0, building (r, s)-divisions and a recursive division tree as follows: it starts with the trivial division D_{I+1} of G_{I+1} as a single region and initializes the recursive division tree T with a single node v_G. Then, for each G_i, it considers each region R of G_{i+1} and builds an (r, s)-division on it (with appropriate values of r and s defined below); each resulting region R' of R (of G_{i+1}) is turned into a region R'' of G_i by expanding every vertex into the at

Algorithm 5.2.1: Generalized Recursive Division Algorithm

Input : An undirected graph $G = (V, E)$ excluding a $\mathbf{K}_\ell$-minor.
Output: A recursive division tree T for G satisfying inequality (5.1) for all r_i's exceeding a constant.

begin
  `// partition and contract the graph recursively`
  let $n := |V|$, $G_0 := G$, $z_0 := 2$, $i := 0$;
  while *the number of vertices in* $G_i > \frac{n}{\log n}$ **do**
    let $G_{i+1} :=$ H-PARTITION(G_i, z_i, ℓ);
    let $z_{i+1} := 14^{z_i^{1/7}}$, $i := i + 1$;
  let $I := i - 1$;
  `// divide the graphs and build recursive division tree`
  let v_G be the root of T;
  let D_{I+1} be the trivial division of G_{I+1} consisting of a single region;
  for $i := I$ *downto* 0 **do**
    for *each region* R *of* D_{i+1} **do**
      let S_R be the boundary-vertices of R in the division D_{i+1};
      let $D_R :=$ DIVIDE(R, S_R, z_i, ℓ);
      for *each region* R' *of* D_R **do**
        expand R' into a region R'' of G_i by expanding every vertex;
        assign each boundary edge to one of the regions it occurs in;
        create a child $v_{R''}$ of v_R in T;
    let D_i be the decomposition of G_i consisting of the regions R'' above;
  `// add the leaves`
  for *each edge* e *of each region* R *of* D_0 **do**
    create a child v_e of v_R in T;
  return T;
end

most $c_0 z_i$ vertices it represents in G_i; afterwards, a child $v_{R''}$ of v_R is added to T. The division D_i is defined to be the decomposition of G_i by the regions R'' obtained this way. Note that a boundary vertex is expanded in multiple regions, creating multiple copies of the edges it expands to; there should be only one copy of these edges and this may be achieved by assigning them to one of these regions arbitrarily.

It remains to specify how exactly and with what parameters the (r, s)-division is built in the iteration above. This is the third place where our algorithm differs from the original. The original algorithm uses a modified version of Frederickson's (r, s)-division algorithm [Fre87], called DIVIDE, as follows: it takes three parameters G, S and r and divides the edges of an n-vertex graph G into at most $c_2(|S|/\sqrt{r} + \frac{n}{r})$ regions, each one having at most r vertices and at most $c_1\sqrt{r}$ boundary vertices, where c_1 and c_2 are constants; a vertex is considered as a boundary vertex if (i) it belongs to more than

one region, or (ii) it belongs to the set S. Internally, the linear-time planar $\mathcal{O}(\sqrt{n})$-size separator of Lipton and Tarjan [LT79] is used to achieve the desired division. We make use of the linear-time $\mathcal{O}(n^{\frac{2}{3}})$-size separator of Reed and Wood [RW09] instead; our DIVIDE procedure takes four parameters G, S, r, and ℓ and has the properties specified in Lemma 5.7 below; as it can be seen in the lemma, a number of constants and exponents are changed. The parameter ℓ is a constant taken to indicate the fixed excluded $\mathbf{K}_\ell$-minor.

In the last phase of the algorithm, the edges of each region R of D_0 are added as children of the node v_R to the recursive division tree T. This completes the description of our generalized algorithm.

5.2.2 Correctness of our Generalized Recursive Division Algorithm

Theorem 5.6. *Algorithm 5.2.1 is a linear-time algorithm that given a $\mathbf{K}_\ell$-minor-free graph G, finds an $(\overline{r}, f)$-recursive division of G that satisfies inequality (5.1) for all r_i exceeding a constant and whose recursive division tree has $\mathcal{O}(n)$ nodes.*

Our proof of the correctness of the algorithm follows very closely the original proof of correctness of Henziger et al.'s recursive division algorithm [HKRS97]. For the sake of completeness, and since a number of subtle details and calculations have to be filled in and replaced at several places, we have included the full proof in this section. Lemma 5.7 shows the correctness of the DIVIDE procedure and is based on the original proof of Frederickson [Fre87]; Lemmas 5.8 – 5.10 step-by-step complete the proof of Theorem 5.6.

Lemma 5.7. *Replacing the planar separator in Frederickson's* DIVIDE *procedure [Fre87] with the separator algorithm of Reed and Wood [RW09] causes the* DIVIDE(G, S, r, ℓ) *procedure to work as follows (where G is a graph with n vertices and excludes $\mathbf{K}_\ell$ as a minor and c_1 and c_2 are constants depending only on ℓ):*

- *it divides G into at most $c_2(|S|/r^{\frac{2}{3}} + \frac{n}{r})$ regions;*
- *each region has at most r vertices;*
- *each region has at most $c_1 r^{\frac{2}{3}}$ boundary vertices, where the vertices in S also count as boundary;*
- *it takes time $\mathcal{O}(n \log n)$.*

Proof. In the following, when we refer to boundary vertices, we mean vertices that belong to more than one region or vertices that belong to the set S. The DIVIDE procedure works as follows: assign weight $\frac{1}{n}$ to each vertex of G and find a $\mathcal{O}(n^{\frac{2}{3}})$-size separator in G and recursively apply the separation algorithm to each region with more than r vertices. Now each region has at most r vertices. While there is a region with more than $c_1 r^{\frac{2}{3}}$ boundary vertices, do the following: if such a region has n' boundary vertices, assign weight $\frac{1}{n'}$ to each of them, assign weight zero to the other vertices of that region, and apply the separator theorem. In the end, all regions will have the desired properties and the algorithm takes time $\mathcal{O}(n \log n)$. It remains to show the bound on the number of the regions.

Consider the division before the regions are further split to enforce the requirement on the number of boundary vertices (i.e. just when we have achieved that each region has size at most r). Let V_B be the set of vertices that are included in more than one region. For a vertex $v \in V_B$, let $b(v)$ be one less than the number of regions that contain v in the division. Let $B(n,r)$ be the total of $b(v)$ over all vertices $v \in V_B$. Thus $B(n,r)$ is the sum of the number of vertices $v \in V_B$ weighted by the count $b(v)$. From the separator theorem in [RW09], we have the following recurrence:

$$\begin{aligned} B(n,r) &\leq d_0 n^{\frac{2}{3}} + B(\alpha n, r) + B((1-\alpha)n, r) \quad \text{for } n > r, \\ B(n,r) &= 0 \qquad\qquad\qquad\quad \text{for } n \leq r \end{aligned} \tag{5.2}$$

where d_0 is a constant and $\frac{1}{2} \leq \alpha \leq \frac{2}{3}$. We claim that

$$B(n,r) \leq d_1 \frac{n}{r^{\frac{1}{3}}} - d_2 n^{\frac{2}{3}} \qquad \text{for } n \geq \frac{r}{3}, \tag{5.3}$$

with some constants d_1 and d_2. The claim can be shown by induction:

As the base of the induction, we consider the cases $\frac{r}{3} \leq n \leq r$. Note that since after splitting a region, each subregion still has at least one-third of the total vertices, it is sufficient to only consider graphs with at least $r/3$ vertices. By choosing $d_1 \geq 3^{\frac{1}{3}} d_2$, we have

$$\frac{d_1 n}{r^{\frac{1}{3}}} \geq \frac{3^{\frac{1}{3}} d_2 n}{3^{\frac{1}{3}} n^{\frac{1}{3}}} = d_2 n^{\frac{2}{3}} \Rightarrow \frac{d_1 n}{r^{\frac{1}{3}}} - d_2 n^{\frac{2}{3}} \geq 0 = B(n,r)\,. \tag{5.4}$$

For the inductive step, i.e. for $n > r$, we have

$$\begin{aligned} B(n,r) &\leq d_0 n^{\frac{2}{3}} + d_1 \frac{\alpha n}{r^{\frac{1}{3}}} - d_2 \alpha^{\frac{2}{3}} n^{\frac{2}{3}} + d_1 \frac{(1-\alpha)n}{r^{\frac{1}{3}}} - d_2 (1-\alpha)^{\frac{2}{3}} n^{\frac{2}{3}} \\ &= d_1 \frac{n}{r^{\frac{1}{3}}} + n^{\frac{2}{3}} (d_0 - d_2 \alpha^{\frac{2}{3}} - d_2 (1-\alpha)^{\frac{2}{3}}) \\ &\leq d_1 \frac{n}{r^{\frac{1}{3}}} - d_2 n^{\frac{2}{3}} \end{aligned} \tag{5.5}$$

if we choose $d_2 \leq d_2 \alpha^{\frac{2}{3}} + d_2 (1-\alpha)^{\frac{2}{3}} - d_0$. This can be achieved by setting $d_2 = 5 d_0 \geq \frac{d_0}{\alpha^{\frac{2}{3}} + (1-\alpha)^{\frac{2}{3}} - 1}$.

In particular, we have shown so far that $B(n,r) = \mathcal{O}(n/r^{\frac{1}{3}})$. The sum of the number of vertices in each region is $n + B(n,r) = n + \mathcal{O}(n/r^{\frac{1}{3}})$ and each region has $\Theta(r)$ vertices, so the number of regions we have so far is $\Theta(n/r)$.

Let t_i be the number of regions with i boundary vertices (recall that in our definition, the set of boundary vertices is $V_B \cup S$). We have

$$\sum_i i t_i = \sum_{v \in V_B} (b(v) + 1) + |S \setminus V_B| < 2B(n,r) + |S| = \mathcal{O}(n/r^{\frac{1}{3}} + |S|). \tag{5.6}$$

Let $s(i)$ be an upper bound on the number of splits that have to be applied to a graph

with at most r vertices and i boundary vertices, until each of its regions has at most $c_1 r^{\frac{2}{3}}$ boundary vertices, for a constant c_1 to be determined. We have that

$$\begin{aligned} s(i) \leq s(\alpha i + d_0 r^{\frac{2}{3}}) + s((1-\alpha)i + d_0 r^{\frac{2}{3}}) + 1 \quad & \text{for } i > c_1 r^{\frac{2}{3}} \\ s(i) = 0 \qquad & \text{for } i \leq c_1 r^{\frac{2}{3}} \end{aligned} \tag{5.7}$$

where $\frac{1}{2} \leq \alpha \leq \frac{2}{3}$. We claim that

$$s(i) \leq \frac{d_3 i}{c_1 r^{\frac{2}{3}}} - \frac{2 d_0 d_3}{c_1} - 1 \qquad \text{for } i \geq \frac{c_1 r^{\frac{2}{3}}}{3} \tag{5.8}$$

for some constant d_3. We prove our claim by induction. Like in the previous induction, for the base case we may assume $\frac{c_1 r^{\frac{2}{3}}}{3} \leq i \leq c_1 r^{\frac{2}{3}}$. By choosing $d_3 = 12$ and $c_1 = 8 d_0$, we have

$$\frac{d_3 i}{c_1 r^{\frac{2}{3}}} - \frac{2 d_0 d_3}{c_1} - 1 \geq 12 \cdot \frac{1}{3} - \frac{24 d_0}{8 d_0} - 1 = 0 = s(i)\,. \tag{5.9}$$

For the inductive step with $i > c_1 r^{\frac{2}{3}}$, note that $\alpha i + d_0 r^{\frac{2}{3}} \leq \frac{2}{3} i + \frac{d_0}{c_1} i \leq (\frac{2}{3} + \frac{1}{8}) i < i$. The same way, we have $(1-\alpha) i + d_0 r^{\frac{2}{3}} < i$. So, we may apply the induction hypothesis to (5.7) and a straightforward calculation will prove our claim.

We have shown that for a region with i boundary vertices, where $i > c_1 r^{\frac{2}{3}}$, at most $\frac{d_3 i}{c_1 r^{\frac{2}{3}}}$ splits need be done for some constants c_1 and d_3. This will result in at most $d_0 r^{\frac{2}{3}}$ new boundary vertices per split and a total of at most $d_3 i / (c_1 r^{\frac{2}{3}})$ new regions. Thus the total number of new boundary vertices is at most

$$\sum_i (d_0 r^{\frac{2}{3}})(d_3 i / (c_1 r^{\frac{2}{3}})) t_i \leq \frac{d_0 d_3}{c_1} \sum_i i t_i = \mathcal{O}(n / r^{\frac{1}{3}} + |S|)\,. \tag{5.10}$$

The number of new regions is at most

$$\sum_i (d_3 i / (c_1 r^{\frac{2}{3}})) t_i = \frac{d_3}{c_1 r^{\frac{2}{3}}} \mathcal{O}(n / r^{\frac{1}{3}} + |S|) = \mathcal{O}(n/r + |S| / r^{\frac{2}{3}})\,. \tag{5.11}$$

□

Recall that we start with the graph $G_0 = G$ and repeatedly apply the procedure H-Partition to each G_i to obtain G_{i+1}. For each i, let n_i denote the number of vertices of G_i. Afterwards we work our way back from G_{I+1} to G_0 and obtain a division D_i on each G_i. Let k_i denote the number of regions of D_i. Note that the recursive division tree T has depth $I + 1$.

The following proof has four parts. First, we show that each region of the division D_i has at most $\mathcal{O}(z_i^2)$ vertices and at most $\mathcal{O}(z_i^{\frac{5}{3}})$ boundary vertices. Second, we show that the number k_i of regions is $\mathcal{O}(n_i / z_i^2)$. Third, we show that Algorithm 5.2.1 takes linear time and finally, we show that the division fulfills inequality (5.1).

For notational convenience, let $z_{I+1} = \sqrt{n_{I+1}}$, so the single region of the division D_{I+1} of G_{I+1} has z_{I+1}^2 vertices. Consider iteration $i \leq I$ in the second phase of the algorithm. By the correctness of DIVIDE, the decomposition D_R of a region of D_{i+1} consists of regions R' of size at most z_i. By the correctness of H-PARTITION, each vertex of G_{i+1} expands to at most $c_0 z_i$ vertices of G_i. Hence, each region R'' obtained from R' by expanding its vertices has size at most $c_0 z_i^2$. Similarly, each region R' has at most $c_1 z_i^{\frac{2}{3}}$ boundary vertices by the correctness of DIVIDE, so the corresponding region R'' has at most $c_0 c_1 z_i^{\frac{5}{3}}$ boundary vertices.

Lemma 5.8. *The number k_i of regions in the division D_i is $\mathcal{O}(n_i/z_i^2)$.*

Proof. We show by reverse induction on i that $k_i \leq c_3 n_i/z_i^2$ for all $i \geq i_0$, where i_0 and c_3 are constants to be determined. For the base case, we have $k_{I+1} = 1$.

Consider iteration $i \leq I$ in the second phase, and suppose $i \geq i_0$. The regions of D_i are obtained by subdividing the k_{i+1} regions comprising the division of G_{i+1}. Since $n_{i+1} \leq n_i/z_i$ and $z_{i+1}^2 \geq z_i$, we have by the induction hypothesis that

$$k_{i+1} \leq c_3 n_{i+1}/z_{i+1}^2 \leq c_3 n_i/z_i^2 \,. \tag{5.12}$$

Each region R of the division of G_{i+1} has $|S_R| \leq c_0 c_1 z_{i+1}^{\frac{5}{3}}$ boundary vertices. Summing over all regions R in D_{i+1}, we obtain

$$\begin{aligned}\sum_R n_R &= \sum_R (\#\text{ of non-boundary vertices} + \#\text{ of boundary vertices}) \\ &\leq n_{i+1} + \sum_R c_0 c_1 z_{i+1}^{\frac{5}{3}} \\ &\leq n_{i+1} + c_0 c_1 k_{i+1} z_{i+1}^{\frac{5}{3}} \,.\end{aligned} \tag{5.13}$$

For each region R, by correctness of DIVIDE, the number of subregions into which R is divided is at most $c_2(|S_R|/z_i^{\frac{2}{3}} + n_R/z_i)$, which is in turn at most $c_2(c_0 c_1 z_{i+1}^{\frac{5}{3}}/z_i^{\frac{2}{3}} + n_R/z_i)$. Summing over all such regions R and using (5.13) and (5.12), we infer that the total number of subregions is at most

$$\begin{aligned}\sum_R c_2(c_0 c_1 z_{i+1}^{\frac{5}{3}}/z_i^{\frac{2}{3}} + n_R/z_i) &= c_0 c_1 c_2 k_{i+1} z_{i+1}^{\frac{5}{3}}/z_i^{\frac{2}{3}} + c_2 \sum_R n_R/z_i \\ &\leq c_0 c_1 c_2 k_{i+1} z_{i+1}^{\frac{5}{3}}/z_i^{\frac{2}{3}} + c_2(n_{i+1} + c_0 c_1 k_{i+1} z_{i+1}^{\frac{5}{3}})/z_i \\ &\leq c_0 c_1 c_2 (\frac{c_3 n_{i+1}}{z_{i+1}^2}) z_{i+1}^{\frac{5}{3}}/z_i^{\frac{2}{3}} + c_2 n_{i+1}/z_i + c_0 c_1 c_2 (\frac{c_3 n_{i+1}}{z_{i+1}^2}) z_{i+1}^{\frac{5}{3}}/z_i \\ &\leq c_0 c_1 c_2 c_3 n_{i+1}/(z_i^{\frac{2}{3}} z_{i+1}^{\frac{1}{3}}) + c_2 n_{i+1}/z_i + c_0 c_1 c_2 c_3 n_{i+1}/(z_i z_{i+1}^{\frac{1}{3}}) \\ &\leq c_0 c_1 c_2 c_3 n_i/(z_i^{\frac{5}{3}} z_{i+1}^{\frac{1}{3}}) + c_2 n_i/z_i^2 + c_0 c_1 c_2 c_3 n_i/(z_i^2 z_{i+1}^{\frac{1}{3}}) \,,\end{aligned} \tag{5.14}$$

where in the last line we use the fact that $n_{i+1} \leq n_i/z_i$. We have obtained an upper

bound on the total number of subregions into which the regions of D_{i+1} are divided. Each subregion becomes a region of D_i. Thus we have in fact bounded k_i, the number of regions of D_i. To complete the induction step, we show that each of the three terms in (5.14) is bounded by $c_3 n_i/3z_i^2$.

The second term, $c_2 n_i/z_i^2$, is bounded by $c_3 n_i/3z_i^2$ if we choose $c_3 \geq 3c_2$. The third term is smaller than the first term. As for the first term, recall that $z_{i+1} = 14^{z_i^{1/7}}$. For sufficiently large choice of i_0, we can ensure that $i \geq i_0$ implies $z_{i+1}^{\frac{1}{3}} \geq 3c_0c_1c_2/z_i^{\frac{2}{3}}$. Thus the first term is also bounded as desired.

We conclude that $k_i \leq c_3 n_i/z_i^2$, completing the induction step. We have shown this inequality holds for all $i \geq i_0$. As for $i < i_0$, clearly $k_i \leq (z_i^2)n_i/z_i^2 \leq (z_{i_0}^2)n_i/z_i^2$. Thus by choosing c_3 to exceed the constant $z_{i_0}^2$, we obtain the lemma for every i. □

Lemma 5.9. *The algorithm runs in linear time.*

Proof. The time required to form the graphs $G_1, G_2, \ldots, G_{I+1}$ is $\mathcal{O}(\sum_i n/z_i)$, which is $\mathcal{O}(n)$. For $i \leq I$, the time to apply DIVIDE to a region R of G_{i+1} with n_R vertices is $\mathcal{O}(n_R \log n_R)$. Each such region has $\mathcal{O}(z_{i+1}^2)$ vertices, so the time is $\mathcal{O}(n_R \log z_{i+1})$. Summed over all regions R, we get $\sum_R \mathcal{O}(n_R \log z_{i+1}) = \mathcal{O}(n_{i+1} \log z_{i+1})$. The time to obtain the induced division of G_i is $\mathcal{O}(n_i)$. Thus the time to obtain divisions of all the G_i's is $\sum_i \mathcal{O}(n_{i+1} \log z_{i+1})$. Since $n_{i+1} \leq n_i/z_i \leq n/z_i$ and $\log z_{i+1} = \mathcal{O}(z_i^{\frac{1}{7}})$, the sum is $\mathcal{O}(n)$. □

Lemma 5.10. *The recursive division obtained by Algorithm 5.2.1 satisfies inequality (5.1).*

Proof. First, note that combining the inequalities $n_{i+1} \leq n_i/z_i$, we obtain

$$n_i \leq n/\prod_{j<i} z_j \,. \tag{5.15}$$

Note moreover that each vertex of G_i expands to at most $\prod_{j<i} c_0 z_j$ vertices of G.

Consider the division D_i of G_i, and the division it induces on G. The division D_i consists of $\mathcal{O}(n_i/z_i^2)$ regions, each having $\mathcal{O}(z_i^2)$ vertices and $\mathcal{O}(z_i^{\frac{5}{3}})$ boundary vertices. This induces $\mathcal{O}(n_i/z_i^2)$ regions in G, each consisting of $\mathcal{O}(z_i^2 \prod_{j<i} c_0 z_j)$ vertices and $\mathcal{O}(z_i^{\frac{5}{3}} \prod_{j<i} c_0 z_j)$ boundary vertices.

Let $r_i = z_i^2 \prod_{j<i} z_j$ and define

$$f(r_i) = z_i^{\frac{5}{3}} \prod_{j<i} c_0 z_j \,. \tag{5.16}$$

Then, by (5.15), the induced division of G has $\mathcal{O}(n/r_i)$ regions each with $\mathcal{O}(r_i c_0^i)$ vertices and $\mathcal{O}(f(r_i))$ boundary vertices. Since $c_0^i = \mathcal{O}(\prod_{j \leq i} z_j)$, we get that the number of

vertices per region is $\mathcal{O}(r_i^2)$. We have

$$\frac{r_i}{f(r_i)} = \frac{z_i^2}{z_i^{\frac{5}{3}} c_0^{i-1}} = \frac{z_i^{\frac{1}{3}}}{c_0^{i-1}}\,. \tag{5.17}$$

Using the definition of z_i, one can verify that $z_{i-1} = \Theta(log^7 z_i)$ and $\prod_{j<i} z_j = \mathcal{O}(log^8 z_i)$. Hence

$$f(r_{i-1}) = c_0^{i-2} z_{i-1}^{\frac{5}{3}} \prod_{j<i-1} z_j = c_0^{i-2}\mathcal{O}(\log^{\frac{35}{3}} z_i \log^8 \log z_i)\,. \tag{5.18}$$

We also have

$$\log r_{i+1} = \log(z_{i+1}^2 \prod_{j\le i} z_j) = \mathcal{O}(\log(z_{i+1}^2 \log^8 z_{i+1})) = \mathcal{O}(\log z_{i+1}) = \mathcal{O}(z_i^{\frac{1}{7}}) \tag{5.19}$$

and consequently $\sum_{j=1}^{i+1} \log r_j = \mathcal{O}(z_i^{\frac{1}{7}})$. For a sufficiently large constant i_0, we have for all $i \ge i_0$,

$$\begin{aligned} 8^i f(r_{i-1}) \log r_{i+1} (\sum_{j=1}^{i+1} \log r_j) &\le 8^i c_0^{i-2}\mathcal{O}(\log^{\frac{35}{3}} z_i \log^8 z_i)\mathcal{O}(z_i^{\frac{1}{7}})\mathcal{O}(z_i^{\frac{1}{7}}) \\ &= 8^i c_0^{i-2}\mathcal{O}(z_i^{\frac{2}{7}} \log^{20} z_i) \le \frac{z_i^{\frac{1}{3}}}{c_0^{i-1}} = \frac{r_i}{f(r_i)}\,, \end{aligned} \tag{5.20}$$

since the z_i's grow much faster than any exponential function having a constant in the base; specifically, we can see below that $z_i^{\frac{1}{21}} \ge g_0^i \log^{20} z_i$ for any constant $g_0 \ge 0$ if i is larger than a constant:

$$\begin{aligned} \frac{1}{21}\log z_i \ge i \log g_0 + 20 \log\log z_i &\Leftrightarrow \frac{1}{21}\log 14^{z_{i-1}^{\frac{1}{7}}} \ge i \log g_0 + 20 \log\log 14^{z_{i-1}^{\frac{1}{7}}} \\ &\Leftrightarrow z_{i-1}^{\frac{1}{7}} \ge g_1 i + g_2 \log z_{i-1}^{\frac{1}{7}} + g_3\,, \end{aligned} \tag{5.21}$$

for some constants g_1, g_2, and g_3. And the last inequality is true if i is large enough, since $z_i^{\frac{1}{7}}$ grows much faster than i. So, inequality (5.1) is fulfilled for all r_i exceeding the constant r_{i_0}. □

5.2.3 Establishing The Degree Requirement

After having computed a recursive division, we still have to transform the graph to have maximum degree 3; otherwise, Theorem 5.3 can not be applied, see Subsection 5.1.4. We can achieve this, using our recursive division, by the following lemma. Note that according to Proposition 5.5. the resulting graph might not be $\mathbf{K}_\ell$-minor-free but it will still serve our purpose of finding shortest paths in linear time since it is now accompanied by a recursive division satisfying inequality (5.1).

Figure 5.4: (a) a given graph with 3 regions indicated by different line styles and colors and red boundary vertices; (b) the transformed graph in which every vertex has degree at most 3; the number of boundary vertices of each region has exactly doubled.

Lemma 5.11. *Let G be an edge-weighted directed graph excluding a fixed minor and let T be a recursive division tree representing an $(\overline{r}, f)$-recursive division of G. Then one can replace every vertex of G with a zero-weight cycle to obtain a graph G' and at the same time modify T into a tree T', so that G' has in-/outdegree at most 2 and T' represents an $(\overline{r}, f)$-recursive division of G'. This modification takes linear time.*

Proof. Recall that the leaves of T represent the edges of G and that internal nodes of T correspond to regions of G, namely, the region induced by all the leaves in the subtree of that node. We modify G and T at the same time. First, for every vertex v of G with degree $\deg(v)$ (the sum of the indegree and outdegree), we add new vertices $v_1, \ldots, v_{\deg(v)}$ to G. We do an in-order traversal of T and for every leaf of T representing an edge $e = vw$ of G, we do the following: let e be the ith edge of v and the jth edge of w that we encounter. We change the endpoints of e to be the vertices v_i and w_j and add two new zero-weight edges v_iv_{i+1} and w_jw_{j+1} as siblings of e to T (if $i = \deg(v)$, we use $v_{\deg(v)}v_1$ instead; same for w). This way, every vertex v of G is replaced by a zero-weight cycle $(v_1, \ldots, v_{\deg(v)})$ (see Figure 5.4). The original vertices of G will become isolated and can be removed. We call the resulting graph G' and the modified recursive division tree T'. Note that since T has size $\mathcal{O}(n)$, this procedure takes only linear time. Also note that we only added new leaves to T, and thus the internal nodes of T and T' correspond one-to-one to each other.

Now consider an internal node q' of T'. It represents a region R' of G' and corresponds to a node q of T, representing a region R of G. R has $r^{\mathcal{O}(1)}$ vertices and $\mathcal{O}(f(r))$ boundary-vertices. The number of edges of R' is at most three times as large as in R and the number of vertices is proportional to the number of edges of R. But R is a subgraph of G and excludes the same fixed minor, and thus the number of its edges is linear in the number of its vertices. Hence, R' still has $r^{\mathcal{O}(1)}$ vertices and edges. Also, since R is represented by the subtree rooted at q, its edges were traversed in order while building T' and G'. So, every vertex v in R is replaced by a path $v_i, v_{i+1}, \ldots, v_j$ with $1 \leq i \leq j \leq \deg(v)$ in R'. Thus, if v is a boundary vertex of R, we have v_i and v_j as

boundary vertices of R' instead. So R' has at most twice as many boundary vertices as R, i.e. still $\mathcal{O}(f(r))$ (see Figure 5.4). So, T' represents an $(\overline{r}, f)$-recursive division of G'. □

Proof of Theorem 5.4. Note that up to the choice of the start- and endvertex inside the zero-weight cycles of G', shortest paths in G and G' correspond one-to-one to each other. G' fulfills all the requirements of Theorem 5.3 and combining this with Theorem 5.6, and Lemma 5.11, we obtain our main theorem, namely, Theorem 5.4. □

5.3 Steiner Tree Approximation in Linear Time on H-Minor-Free Graphs

Let $G = (V, E)$ be a given graph and $R \subseteq V$ a given set of terminals. Recall from Section 1.2 that Mehlhorn's algorithm for finding a 2-approximate Steiner tree [Meh88] involves the following steps:

(i) decomposing the graph into Voronoi regions using one shortest-paths computation;
(ii) using these regions to build the reduced distance network $N_D^\star$ of the terminals;
(iii) finding the minimum spanning tree (MST) of $N_D^\star$;
(iv) returning the union of the shortest paths in G that correspond to the MST.

We show how to implement this algorithm in linear time on proper minor-closed graph classes using Theorem 5.4 and the observation that Mehlhorn's distance network is a minor of the input graph. In what follows, we prove the following theorem:

Theorem 5.12. *There is a linear-time algorithm that calculates a 2-approximation for the Steiner minimum tree problem in any proper minor-closed class of graphs.*

We first show how to find the Voronoi regions in linear time. In graphs excluding a fixed $\mathbf{K}_\ell$-minor, we observe that the graph with an added super-source will exclude $\mathbf{K}_{\ell+1}$; so, Theorem 5.4 applies and shortest paths can be calculated in linear time. Alternatively, using a similar method as in Subsection 5.2.3, one can first find a recursive division of G and then add the super-source and its edges to G and to the recursive division. This could result in much better constants in the running time of the algorithm, especially for planar graphs. We get

Lemma 5.13. *For a graph G excluding a fixed minor and having nonnegative edge-weights and a given set of terminals R in G, the Voronoi regions of G with respect to R can be determined in linear time.*

Corollary 5.14. *In a proper minor-closed class of graphs, the distance network $N_D^\star$ can be calculated in linear time for any given set of terminals in a given graph from the class.*

The next step of Mehlhorn's algorithm is to calculate the minimum spanning tree of $N_D^\star$. But notice that $N_D^\star$ is obtained by contracting the Voronoi regions of the graph (which are connected) and removing loops and parallel edges, i.e.

Observation 5.15. *For a given graph G and a set of terminals, the distance network $N_D^\star$ is a minor of G.*

Thus, $N_D^\star$ belongs to the same proper class of minor-closed graphs as G and one can apply the linear-time minimum spanning tree algorithm of Mares [Mar04]. When we are dealing with planar graphs, the algorithm of Cheriton and Tarjan can be used [CT76]. As mentioned before, the last step of Mehlhorn's algorithm is to replace the edges of $N_D^\star$ with the corresponding paths from G and this can clearly be done in linear time. Hence, Theorem 5.12 is proven. □

5.4 Conclusion and Outlook

We showed how to generalize the linear-time shortest paths algorithm of Henzinger et al. [HKRS97] from planar graphs to H-minor-free graphs. We argued that a straightforward generalization of the algorithm cannot work because in general H-minor-free graphs, the shortest-paths problem cannot be reduced to graphs of bounded degree by splitting vertices while maintaining the exclusion of a fixed minor. The main issue is that we do not have an order on the neighbors of each vertex as we have in embedded graphs; and so the splitting of vertices becomes problematic. Our main idea was to first find a recursive division and then use that division to define a suitable ordering for the splitting of vertices. To this end, we had to generalize the recursive division algorithm of Henzinger et al. to work on H-minor-free graphs of unbounded degree. We achieved this by using the concept of a connected H-partition and fast separation in H-minor-free graphs as recently given by Reed and Wood [RW09].

As an application, we showed how to obtain a 2-approximation for Steiner tree in linear time in H-minor-free graphs. A linear time constant-factor approximation algorithm for this problem was previously not known on any nontrivial graph classes.

The most important open question that remains is whether a linear time shortest paths algorithm exists for general graphs; and if not, to identify the classes of graphs for which this is possible. Similarly, it would be interesting to find out which classes of graphs admit a linear-time constant-factor approximation of the Steiner tree problem.

6 Faster PTASes and FPT-Algorithms on (Odd-)H-Minor-Free Graphs

One of the seminal results in algorithmic graph theory is arguably Baker's approach for designing polynomial-time approximation schemes for a wide range of problems on planar graphs [Bak94]. Ever since its discovery, it has been applied and generalized in various ways, see e.g. [Epp00a, FG01, Gro03, AFN04, DHK05, DHM07, Kle08, BKM09]. The essence of the idea is the following: for any given t, one can partition a planar graph into t parts, so that removing any one of the parts results in a graph of bounded treewidth. Now, to obtain a PTAS, we observe that if t is appropriately chosen, there must exist a part that contains at most an ε-fraction of an optimal solution; this can often be combined with the solution in the remainder of the graph to obtain a $(1+\varepsilon)$-approximation.

H-minor-free graphs have gained significant attention in the past two decades, especially due to Robertson and Seymour's graph minor theory. As already mentioned, these classes include, e.g. planar graphs, bounded-genus graphs, linklessly embeddable graphs and apex graphs. Using the deep Robertson-Seymour (RS-) decomposition theorem [RS03], Grohe [Gro03] generalized Baker's technique to H-minor-free graphs and Demaine et al. [DHK05] showed the partitioning theorem mentioned above for all these graph classes. However, both their methods result in algorithms with running time $\mathcal{O}(n^{f(|H|)})$, for some computable function f; since H is assumed to be fixed, this is considered polynomial.

Improving Baker's Decomposition We provide the first algorithm for Baker's decomposition of H-minor-free graphs running in time $\mathcal{O}(g(|H|)n^{\mathcal{O}(1)})$, for some computable function g. This is a significant acceleration of the previous results, especially considering the fact that the constants in graph minor theory, such as the functions f, g above, are usually huge. This immediately implies similar improvements on all the consequences of this algorithm, especially *all* the generic approximation algorithms and schemes in [Gro03, DHK05] and Baker's original problems [Bak94]. In particular, we obtain the first 2-approximation for Coloring H-minor-free graphs in the given time bound and the first PTAS for Independent Set, Minimum Color Sum, Max-Cut, Maximum P-Matching, and Dominating Set on these graph classes while avoiding $|H|$ in the exponent of n in their running time. Our main idea is derived from Dawar et al.'s approach [DGK07] of finding a certain tree decomposition of H-minor-free graphs that is more tractable than the RS-decomposition. In the language of parameterized complexity, our result above shows that partitioning H-minor-free graphs in the described way is in FPT when parameterized by $|H|$.

Guess and Conquer Recall that a parameterized problem of size n with parameter k is in FPT if it can be solved in time $\mathcal{O}(f(k)n^{\mathcal{O}(1)})$, for some computable function f. Once a problem is shown to be FPT, the challenge is to provide algorithms that have the smallest dependence on the parameter k, i.e. make the function f in the running time as small as possible. It is especially desirable to obtain *subexponential* functions and thus provide particularly fast algorithms. Whereas this is often not possible in general graphs, a plethora of results exist that show the existence of such algorithms on restricted graph classes, such as H-minor-free graphs. Perhaps the most general technique to obtain subexponential parameterized algorithms on these graph classes is the theory of *bidimensionality* [DFHT05] that captures almost all known results of this type on H-minor-free graphs. Still, this theory does not apply to a number of prominent problems, such as k-Steiner Tree, Connected k-Dominating Set, and Directed k-Path.

In this work, we provide a new framework, that we call *guess and conquer*, to obtain (nearly) subexponential parameterized algorithms on H-minor-free graphs for an abundant number of parameterized problems. Whenever the problem at hand admits a minor-monotone subexponential kernel, our method results indeed in a subexponential algorithm; otherwise, we obtain an algorithm with a running time $\mathcal{O}(2^{\mathcal{O}_H(\sqrt{k \log n})}n^{\mathcal{O}(1)}) = \inf_{0<\varepsilon\leq 1} \mathcal{O}((1+\varepsilon)^k + n^{\mathcal{O}_H(1/\varepsilon)})$ which we call *nearly* subexponential. Note that if $k = \mathcal{O}(\log n)$, our running time is fully polynomial in the input and if $k = \omega(\log n)$, it is subexponential FPT in k. Hence, except for a "small range" of possible parameter values, we have a subexponential FPT algorithm. In fact, we show that the problems we consider, admit a minor-monotone subexponential kernel on H-minor-free graphs if and only if they admit a subexponential FPT algorithm on these graph classes. Note that in general graphs, even a linear kernel results only in an exponential FPT-algorithm.

Our technique applies in particular to the Connected k-Dominating Set problem and k-Steiner Tree (at least) in bounded-genus graphs and Directed k-Path in all H-minor-free graphs, none of which are known to admit subexponential FPT-algorithms in H-minor-free graphs; for the latter two, such algorithms are not even known for planar graphs.

At the time of preparation of this thesis, we became aware that Dorn et al. [DFL$^+$10] recently and independently obtained similar nearly subexponential algorithms for some problems, albeit only on apex-minor-free graphs – whereas our techniques apply to general H-minor-free graphs. The focus of their work is on directed graph problems and in particular, they obtain a nearly subexponential FPT-algorithm for Directed k-Path in apex-minor-free graphs (furthermore, they obtain a number of subexponential FPT-algorithms for problems that we do not consider in this work). The second method they present in their paper is indeed similar to what we call guess and conquer in this work but it is only formulated for a specific problem, not in general terms as we do, and also not for the wide range of problems we consider; in particular, our technique for domination and covering problems is completely new. Additionally, they present their method only for problems with a polynomial kernel whereas we also obtain nearly subexponential algorithms on problems without such a kernel. Furthermore, even on problems with a

kernel we obtain faster algorithms, having a running time of $\mathcal{O}(2^{\mathcal{O}(\sqrt{k \log k})} n^{\mathcal{O}(1)})$ instead of $\mathcal{O}(2^{\mathcal{O}(\sqrt{k} \log k)} n^{\mathcal{O}(1)})$ of [DFL+10].

Odd-Minor-Free Graphs The class of odd-minor-free graphs has attained extensive attention in the graph theory literature [Gue01, GGG+04] and recently, in theoretical computer science [KM08, DHK10, KLR10]. They are strictly more general than H-minor-free graphs as they include, for example, all bipartite graphs and may contain a quadratic number of edges. In addition to their role in graph minor theory and structural graph theory, they bear important connections to the Max-Cut problem [Gue01] and Hadwiger's conjecture [Had43, KM08]. We refer to the work of Demaine et al. [DHK10] for a more thorough introduction to odd-minor-free graphs and their significance.

Demaine et al. [DHK10] prove a decomposition theorem for odd-H-minor-free graphs that is similar to the RS-decomposition of H-minor-free graphs and present an $\mathcal{O}(n^{f(|H|)})$ algorithm to compute such a decomposition. From this, they derive a Baker-style decomposition of odd-minor-free graphs into two graphs of bounded treewidth. We identify an intermediate decomposition implicit in [DHK10] that is computable in FPT-time and proves to be very useful algorithmically: on one hand, we deduce the Baker-style decomposition into two parts and a number of 2-approximation algorithms (most notably for Coloring) in FPT-time as a corollary; on the other hand, we can answer a question that is posed several times by Demaine et al. in [DHK10], affirmatively: namely, whether the PTASes and subexponential FPT-algorithms for Vertex Cover and Independent Set can be generalized from H-minor-free graphs to odd-minor-free graphs. We show how to obtain such algorithms by introducing a novel dynamic programming technique on odd-minor-free graphs based on solving a certain weighted version of the considered problems in bipartite graphs. These are the first PTASes and subexponential FPT-algorithms developed on odd-minor-free graphs.

Notation and Basic Definitions

We review and collect some relevant concepts from parameterized complexity and graph theory that were mostly not discussed so far. Recall that H-minor-free graphs have bounded average degree (depending on $|H|$), i.e. they fulfill $m = \mathcal{O}_H(n)$ [Mad67]. We use the notation $\mathcal{O}_H$ to denote that the constants hidden in the big-$\mathcal{O}$ depend on $|H|$; this is necessary since in graph minor theory, the exact dependence is often not known.

We denote the standard parameterization of a problem Π by k-Π, i.e. the problem Π parameterized by the solution size k, which is usually the number of vertices or edges in the solution; this applies in particular to k-Steiner Tree.

The Classes SUBEPT and SUBEPT$^+$ Recall that the classes EPT and SUBEPT are defined to be the bounded parameterized complexity classes $2^{\mathcal{O}(k)}$-FPT, and $2^{o^{\text{eff}}(k)}$-FPT. A problem is *subexponential fixed-parameter tractable* if it is in SUBEPT. Observe that if a problem is in SUBEPT then there exists an algorithm for the problem, so that for any fixed $\alpha > 0$ the algorithm runs in time $\mathcal{O}(2^{\alpha k} n^{\mathcal{O}(1)})$. We define *nearly* subexponential parameterized algorithms and problems as follows.

Definition 6.1. A parameterized problem k-Π is said to be in SUBEPT^+ if it can be solved by an algorithm $\mathcal{A}$ such that for any fixed $\alpha > 0$, the running time of $\mathcal{A}$ is bounded by $\mathcal{O}(2^{\alpha k} n^{\mathcal{O}(1/\alpha)})$. In this case, $\mathcal{A}$ is called a *nearly subexponential time* algorithm.

Observe that we require a *single* (uniform) algorithm to have this property for the considered problem. Clearly, $\mathsf{SUBEPT} \subseteq \mathsf{SUBEPT}^+ \subseteq \mathsf{EPT}$. Note that the *non-uniform exponential time hypothesis* (ETH) implies that $\mathsf{SUBEPT}^+ \neq \mathsf{EPT}$.

Note that when we claim that a certain problem k-Π is in SUBEPT or SUBEPT^+ on (odd-)H-minor-free graphs, we mean it is subexponential in k; whenever we would like to talk about $|H|$ as the parameter, we make it explicit.

Odd Minors Recall that a *model* of H in G is a map that assigns to every vertex v of H, a connected subtree T_v of G such that the images of the vertices of H are all disjoint in G and there is an edge between them if there is an edge between the corresponding vertices in H. A graph H is a minor of G if and only if G contains a model of H.

Definition 6.2. A graph H is an *odd-minor* of a graph G if H is a minor of G, and additionally the vertices of the trees in the model of H in G can be 2-colored in such a way that

(i) the edges of each tree T_v are bichromatic; and
(ii) every edge e_G in G that connects two trees T_u and T_v and corresponds to an edge $e_H = uv$ of H is monochromatic.

A graph is *odd-H-minor-free* if it excludes H as an odd minor.

For example, bipartite graphs are odd-K_3-minor-free.

Tree Decompositions and Dynamic Programming We denote a tree decomposition of a graph G by a pair $(T, \mathcal{B})$, where T is a tree and $\mathcal{B} = \{B_u | u \in V(T)\}$ is the family of the bags of the tree decomposition. For a vertex $v \in V(G)$, we let T_v be the connected subtree of T whose bags contain v. The *adhesion* of a tree decomposition is defined as $\max\{|B_u \cap B_t| \mid \{u,t\} \in E_T\}$ (see Figure 6.1 (a)). We denote the treewidth of a graph G by $\mathrm{tw}(G)$.

Many NP-hard optimization problems become fixed-parameter tractable when parameterized by the treewidth of the instance, by using dynamic programming on a given tree decomposition. The most well-known result in this area is Courcelle's theorem [Cou90] stating that any problem definable in monadic second-order logic is in FPT when parameterized by the treewidth and the length of the formula. However, the algorithms obtained by this theorem usually have multiply-exponential dependence on the treewidth of G. In this work, we are interested in algorithms with singly-exponential dependence on the treewidth, i.e. problems that are in EPT when parameterized by treewidth. Several natural problems have long been known to admit such algorithms [AP89, Bod88], and for problems with global connectivity requirement such as Longest Path, Steiner Tree and Connected Dominating Set, EPT-algorithms on (some classes of) H-minor-free graphs were recently given by Dorn et al. [DFT08] by utilizing Catalan structures.

6.1 Partitioning H-Minor-Free Graphs

In [DHK05], Demaine et al. show how to decompose H-minor free graphs into parts, so that upon removal of any part, the problem at hand becomes tractable. In this section, we show how this decomposition can be achieved in FPT-time with $|H|$ as parameter; furthermore, we introduce a refinement of this method.

Theorem 6.3 (Demaine et al. [DHK05])**.** *For every graph H there is a constant c_H such that for any integer $p \geq 1$ and for every H-minor-free graph G, the vertices (edges) of G can be partitioned into p sets such that any $p-1$ of the sets induce a graph of treewidth at most $c_H p$. Furthermore, such a partition can be found in time $n^{\mathcal{O}_H(1)}$.*

The essence of this idea goes back to Baker's approach [Bak94] for polynomial-time approximation schemes on planar graphs. That approach has been applied and generalized in many ways [Epp00a, FG01, Gro03, AFN04, DHK05, DHM07, Kle08, BKM09]. By now, it is considered a standard technique and it is not hard to see that it is true for apex-minor-free graphs: we perform breadth-first search (BFS) and label the BFS-layers periodically by $0, \ldots, p-1$; now by removing the vertices of any one label, the graph falls apart into a number of connected components. If we consider such a connected component C in the BFS-tree then, by *contracting* everything preceding C into a single vertex and *deleting* everything following C in the tree, we obtain a graph of bounded diameter. Such a graph has bounded treewidth because of the bounded local treewidth property of apex-minor-free graphs, and so we obtain a linear time algorithm.

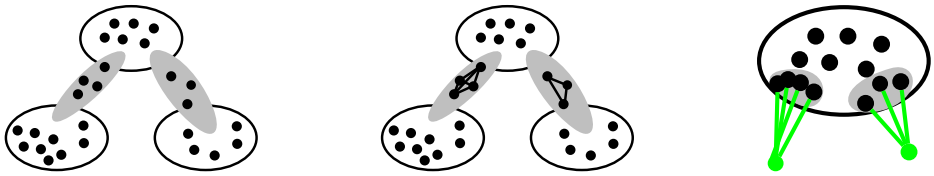
But for general H-minor-free graphs the situation is more complicated; the bounded local treewidth property does not hold for all H-minor free graphs. To overcome this difficulty, Demaine et al. [DHK05] apply the *Robertson-Seymour decomposition (RS-decomposition)* of H-minor-free graphs [RS03], following some ideas of Grohe [Gro03]. They give an algorithm to compute an RS-decomposition of a given H-minor-free graph in time $n^{\mathcal{O}_H(1)}$. For parameter $|H|$, that algorithm is hence not a fixed-parameter algorithm. We now show how the techniques of Grohe [Gro07a] and Dawar et al. [DGK07] can be used to establish fixed-parameter versions and extensions of Theorem 6.3.

6.1.1 The Existence of a Fast Partitioning Algorithm

A key observation to obtain an FPT-version of the partitioning algorithm is that an RS-decomposition is not needed – it suffices to have a tree decomposition of the input graph that fulfills certain properties. To state these properties, we require the following classes of graphs as defined by Grohe [Gro03]:

$$\mathcal{L}(\lambda) = \{G \mid \forall\ H \preceq G \forall\ r \geq 1\ :\ \mathrm{ltw}_r(H) \leq \lambda \cdot r\},$$
$$\mathcal{L}(\lambda, \mu) = \{G \mid \exists\ U \subseteq V(G)\ :\ |U| \leq \mu \text{ and } G - U \in \mathcal{L}(\lambda)\}\ .$$

Since the property of having bounded local treewidth is not inherited when taking minors, we explicitly require it for all minors of G in the definition of $\mathcal{L}(\lambda)$. A graph G in the class $\mathcal{L}(\lambda, \mu)$ may contain a set U of at most μ *apices*, so that by removing



(a) (b) (c)

Figure 6.1: (a) A tree decomposition of adhesion 4; the shaded ovals represent edges of the tree decomposition and the vertices therein are in the intersection of the adjacent bags; (b) building the closure of bags; (c) building the companion of bags; the hat vertices are in green outside the bags.

these apices from G we obtain a graph in $\mathcal{L}(\lambda)$. Note that both of these classes are minor-closed and hence, by the Graph Minor Theorem and the minor-testing algorithm of Robertson and Seymour [RS95, RS04], they can be recognized in time $\mathcal{O}_H(n^3)$.

Given a graph G, consider a tree decomposition $(T, \mathcal{B})$ of G and a bag B in $\mathcal{B}$. The *closure* of B, denoted by $\overline{B}$, is the graph obtained from $G[B]$ by adding some edges so that $B \cap B'$ induces a clique in $\overline{B}$, for every bag $B' \neq B$ (see Figure 6.1 (b)). We say G has a tree decomposition *(strongly) over* a class of graphs $\mathcal{C}$ if there exists a tree decomposition of G so that the closure of each bag is in $\mathcal{C}$. Note that if $\mathcal{C}$ is minor-closed then the class of graphs having a tree decomposition over $\mathcal{C}$ is minor-closed, too. Using the Robertson-Seymour decomposition theorem [RS03], Grohe [Gro03] proved that for every H there exist computable λ, μ, and κ depending only on $|H|$, so that any H-minor-free graph admits a tree decomposition over $\mathcal{L}(\lambda, \mu)$ with *adhesion at most* κ. Later [Gro07a] he observed that such a tree decomposition can be computed in time $\mathcal{O}_H(n^5)$, provided that the excluded minors of the class of graphs having such a decomposition are known. In fact, he proved the *existence* of such an algorithm for all proper minor-closed graph classes without presenting it *explicitly* for any particular class; a common fate when applying the Graph Minor Theorem. Furthermore, this algorithm is *non-uniform* in the sense that for every excluded minor H, we obtain a different algorithm. Nevertheless, based on that decomposition, the proof of Theorem 6.3 can easily be adapted to obtain

Theorem 6.4. *There exists an algorithm that computes a partition as described in Theorem 6.3 in time* $\mathcal{O}_H(n^5)$.

6.1.2 An Explicit FPT-algorithm

The statement of Theorem 6.4 is not quite satisfactory; we would like to *know* and furthermore, have a *uniform* algorithm to compute the desired decomposition. Dawar et al. [DGK07] attacked this problem in the following way: instead of looking at the *closure* of bags in a tree decomposition, they look at the *companion* of the bags: for a bag B with neighbors $B_1, \ldots, B_t$ in a given tree decomposition, we define its companion $\widehat{B}$ as the graph obtained from $G[B]$ by adding new vertices $\widehat{u}_1, \ldots, \widehat{u}_t$ and connecting

$\widehat{u}_i$ to all vertices in the intersection $B \cap B_i$, for $1 \leq i \leq t$. We call $\widehat{u}_1, \ldots, \widehat{u}_t$ the *hat vertices of* $\widehat{B}$ (see Figure 6.1 (c)). Note that the difference between the closure and the companion of a bag is that in the closure, the intersections with neighboring bags form a clique instead of being connected to a hat vertex. We say that a graph G has a tree decomposition *weakly over* a graph class $\mathcal{C}$ if there exists a tree decomposition of G so that the companions of all the bags are in $\mathcal{C}$.

Theorem 6.5 (Dawar et al. [DGK07])**.** *There is an explicit uniform algorithm that, given an H-minor-free graph G, computes a tree decomposition $(T, \mathcal{B})$ of G that is weakly over $\mathcal{L}(\lambda, \mu)$ and has adhesion at most κ, in time $\mathcal{O}_H(n^{\mathcal{O}(1)})$, where λ, μ, and κ are computable functions depending only on $|H|$. Furthermore, the μ apices of the companion of each bag in $\mathcal{B}$ can be computed in the same time bound.*

Note that the λ, μ, and κ in the theorem above are much larger than the ones in the existential version proven by Grohe [Gro03]; but they still depend solely on $|H|$ and are thus acceptable for our purposes. However, in order to adapt the proof of Theorem 6.3 as given in [DHK05] for obtaining an FPT-algorithm, we would need the closure of the bags of the tree decomposition to be in $\mathcal{L}(\lambda, \mu)$. We resolve this issue by using Lemma 6.7 below. First, we need some preparation:

Let G be a graph and let $(T, \mathcal{B})$ be a tree decomposition of G with adhesion at most κ that is weakly over $\mathcal{L}(\lambda, \mu)$, for some κ, λ, and μ, and assume T is rooted at some bag. We say an apex set A of the companion $\widehat{B}$ of a bag $B \in \mathcal{B}$ is *nice*, if

(i) for the parent B' of B in T, we have $B \cap B' \subseteq A$; and
(ii) if $\widehat{u}$ is a hat vertex of $\widehat{B}$ belonging to A, then $N(\widehat{u}) \subseteq A$.

Note that by going from $\mathcal{L}(\lambda, \mu)$ to $\mathcal{L}(\lambda, \mu\kappa + \kappa)$ if necessary, we may assume w.l.o.g that all companions have nice apex sets: simply add the intersection with the parent bag and all the neighbors of included hat vertices to a given apex set. We proceed with our main technical lemmas.[1]

Lemma 6.6. *Let G be an H-minor-free graph and let $(T, \mathcal{B})$ be a tree decomposition of G with adhesion at most κ that is weakly over $\mathcal{L}(\lambda, \mu)$. Consider a bag $B_0 \in \mathcal{B}$ with nice apex set $A \subseteq V(\widehat{B_0})$ and closure $\overline{B}_0$. Define $B := B_0 - A$ and $\overline{B} := \overline{B}_0 - A$. Let $j \geq i \geq 0$ be integers, $r \in B$, and $L^r_{i,j} := \{v \in B \mid i \leq \operatorname{dist}_{\overline{B}}(r, v) \leq j\}$. Then we have $\operatorname{tw}(\overline{B}[L^r_{ij}]) \leq 4\lambda\kappa \cdot (j - i + 1)$.*

Proof. Let $\widehat{U}$ be the set of hat vertices of $\widehat{B}_0$ and $\widehat{B} := \widehat{B}_0 - A$. For $v \in B$, we let $d(v)$ denote $\operatorname{dist}_{\overline{B}}(r, v)$ and define $p := j - i + 1$. For a set $C \subseteq B$, let $\overline{C} := \overline{B}[C]$ denote its closure, and $\widehat{C} := \widehat{B}[C] \cup \widehat{U}_C$ be its companion, where $\widehat{U}_C$ denotes the set of hat vertices in $\widehat{U}$ that have a neighbor in C. Note that $\overline{C}$ is connected if and only if $\widehat{C}$ is connected and that the diameter of $\widehat{C}$ is at most twice that of $\overline{C}$; furthermore, $\widehat{C}$ is disjoint from A because A is nice.

From now on, fix $C := L^r_{ij}$, and let $\overline{C}$ and $\widehat{C}$ denote its closure and companion, respectively. Let R be the set of all $v \in B$ with $d(v) < i$, $\overline{R}$ its closure, and $\widehat{R}$ its

[1] Note that Lemma 6.6 is implicitly assumed by Dawar et al. [DGK07].

companion. Note that $\overline{R}$ and $\widehat{R}$ are connected; but there exists a (possibly empty) set $\widehat{U}' := V(\widehat{C}) \cap V(\widehat{R})$ of hat vertices that are contained in both $\widehat{R}$ and $\widehat{C}$. We claim that $\widehat{R}' := \widehat{R} - \widehat{U}'$ is still connected: to see this, let $\widehat{u} \in \widehat{U}'$ be such a hat vertex and note that $\widehat{u}$ has only neighbors $N_R \subseteq R$ and $N_C \subseteq C$, so that $N_R \cup N_C$ induces a clique in $\overline{B}$. But then, it must be that the vertices $v_R \in N_R$ fulfill $d(v_R) = i - 1$ and the ones $v_C \in N_C$ fulfill $d(v_C) = i$, and therefore the vertices of N_R are connected to r via a path that does not include any of the edges of this clique. Hence $\widehat{R} - \widehat{u}$ is connected.

Now let $Q := C \cup R$, $\overline{Q}$ its closure, and $\widehat{Q}$ its companion. Consider the graph $\widehat{Q}' := \widehat{Q}/E(\widehat{R}')$ obtained by contracting $\widehat{R}'$ in $\widehat{Q}$. Since $\widehat{R}'$ is connected and disjoint from $\widehat{C}$, we observe that $\widehat{Q}'$ is isomorphic to $\widehat{C}$ augmented by a single vertex r' that is connected to all vertices of $v \in C$ with $d(v) = i$ – either by a direct edge or by using a hat vertex from $\widehat{U'}$. Hence, the distance of any vertex $v \in C$ from r' is at most $2p$ in $\widehat{Q}'$, and so the diameter of $\widehat{Q}'$ is at most $4p$. On the other hand, we have $\widehat{Q}' \preceq \widehat{B} \in \mathcal{L}(\lambda)$, and hence the treewidth of $\widehat{Q}'$ is bounded by $4\lambda \cdot p$.

Let $(T', \mathcal{B}')$ be a tree decomposition obtained by (i) considering a tree decomposition $(T_0, \mathcal{B}_0)$ of $\widehat{Q}'$ of width at most $4\lambda \cdot p$; (ii) removing the vertex r' from every bag in $\mathcal{B}_0$; and (iii) replacing every hat vertex in each bag in $\mathcal{B}_0$ by the set of its neighbors. Clearly, $(T', \mathcal{B}')$ is a tree decomposition of $\overline{C}$ and its width is at most $4\lambda\kappa \cdot p$, as desired. □

Lemma 6.7. *Let G be an H-minor-free graph and let $(T, \mathcal{B})$ be a tree decomposition of G with adhesion at most κ that is weakly over $\mathcal{L}(\lambda, \mu)$, where λ, μ, and κ are computable functions depending only on $|H|$. Consider a bag $B_0 \in \mathcal{B}$ with a given nice apex set $A \subseteq V(\widehat{B_0})$. For any integer $p \geq 1$ we can label the vertices and edges of the closure $\overline{B}_0 - A$ by the numbers $\{0, \ldots, p-1\}$, so that the following holds:*

1. *every edge has the label of one of its endpoints;*
2. *every vertex is incident to at most 2 distinct edge-labels;*
3. *the vertices or edges of any $p-1$ labels induce a graph of treewidth at most $4\lambda\kappa \cdot p$;*
4. *there exists an explicit uniform algorithm to find such a labeling in time $\mathcal{O}_H(n^{\mathcal{O}(1)})$.*

Proof. Define $B := B_0 - A$ and $\overline{B} := \overline{B_0} - A$. W.l.o.g. we may assume that $\overline{B}$ is connected, since otherwise we can simply repeat the following procedure for every connected component. We pick an arbitrary vertex r and perform a BFS in $\overline{B}$. We assign the label $\mathrm{lab}(v) = \mathrm{dist}_{\overline{B}}(v, r) \bmod p$ to every vertex $v \in V$; every edge is assigned the label of its endpoint closest to r. Consider a connected component $\overline{C}$ of the graph induced by any $p-1$ vertex or edge labels. Then $\overline{C}$ is a subgraph of $L^r_{i,i+p-1}$, for some $i \geq 0$, and hence, by Lemma 6.6 its treewidth is bounded by $4\lambda\kappa \cdot p$. The other claims are easy to verify. □

Using Theorem 6.5 and Lemma 6.7, it is not hard to extend the proof of Theorem 6.3 in [DHK05] to obtain the following version; on the other hand, this result is implicit in the proof of Theorem 6.9 as given below:

Theorem 6.8. *There exists an explicit uniform algorithm that computes a partition as described in Theorem 6.3 and runs in time $\mathcal{O}_H(n^{\mathcal{O}(1)})$.*

6.1.3 Bounding the Number of Label Incidences

In some of our applications, we need a more specific version of Theorem 6.3; we would like to obtain a partition of the *edges* while still being able to bound the number of parts in which each *vertex* might appear. To this end, we shall bound the number of distinct edge-labels incident to each vertex in an edge-partition of the graph. A closer look at Demaine et al.'s [DHK05] proof of Theorem 6.3 reveals that this number is indeed bounded by $\mathcal{O}_H(1)$; for the sake of completeness, and since our setting is somewhat different, we include a proof in this section.

For two graphs G_1 and G_2 whose intersection $E(G_1) \cap E(G_2)$ induces a clique, we define their *clique-sum* $G_1 \oplus G_2$ as the graph $G_1 \cup G_2$ with any number of edges in the clique $E(G_1) \cap E(G_2)$ deleted. Note that this operation is not well-defined and can have a number of possible outcomes. The notion of a clique-sum plays a central role in graph minor theory and it is well-known that $\mathrm{tw}(G_1 \oplus G_2) \leq \max\{\mathrm{tw}(G_1), \mathrm{tw}(G_2)\}$. A key observation is that when considering two neighboring bags B_1 and B_2 in a tree decomposition of a graph, the graph induced by $B_1 \cup B_2$ is a subgraph of a clique-sum $\overline{B_1} \oplus \overline{B_2}$ of the closure of the bags. We use this observation to prove the following theorem.

Theorem 6.9. *For any fixed graph H there are constants c_H and d_H such that for any integer $p \geq 1$ and every H-minor-free graph G, the edges of G can be partitioned into p parts such that any $p-1$ of the parts induce a graph of treewidth at most $c_H p$ and every vertex appears in at most d_H of the parts. Furthermore, such a partition can be found in explicit uniform* FPT*-time, i.e. $\mathcal{O}_H(n^{\mathcal{O}(1)})$.*

Proof. First, we compute a tree decomposition $(T, \mathcal{B})$, weakly over $\mathcal{L}(\lambda, \mu)$ with adhesion κ as given by Theorem 6.5. We root the tree decomposition at a bag B_0 and let $B_0, \ldots, B_k$ be a pre-order traversal of the bags of the rooted tree decomposition. For $0 \leq i \leq k$, let $G_i := \overline{B}_0 \oplus \cdots \oplus \overline{B}_i$; note that $G = G_k$. For each $i \in \{1, \ldots, k\}$, let C_i be the set of at most κ vertices in the intersection of B_i with its parent bag and $C_0 = \varnothing$; also, let $\widehat{A}_i$ be the set of at most μ apex vertices of the companion $\widehat{B}_i$ and assume w.l.o.g. that $\widehat{A}_i$ is nice, i.e. in particular, $C_i \subseteq \widehat{A}_i$; let $A_i = \widehat{A}_i \cap B_i$. We prove the statement by induction on i, label the vertices and edges simultaneously, and keep the invariant that the label of every edge is equal to the label of one of its endpoints. We say that a label is *incident* to a vertex v if it is the label of v or the label of an edge that is incident to v.

For $G_0 = \overline{B}_0$, we start with the labeling provided by Lemma 6.7. Next, we assign the label 0 to all vertices and edges that are included or have an endpoint in A_0. Since Lemma 6.7 guarantees that every vertex in $B_0 - A_0$ is incident to at most 2 distinct labels, we obtain that in G_0, every vertex is incident to at most 3 distinct labels. Also, the treewidth of any subgraph of G_0 induced by any $p-1$ labels is at most $c_H p$ with $c_H := 4\lambda\kappa + \mu$, as one can simply add all the vertices of A_0 to every bag in a tree decomposition provided by Lemma 6.7.

Now consider G_i for $1 \leq i \leq k$ and let B_j be the parent bag of B_i. We consider the labeling inductively constructed for G_{i-1} and label the vertices and edges of $\overline{B}_i - A_i$ using Lemma 6.7. Since $C_i = B_i \cap B_j = B_i \cap V(G_{i-1}) \subseteq A_i$, we let $V(G_{i-1})$ and all the

edges with both endpoints in $V(G_{i-1})$ keep the labels obtained for G_{i-1} without causing a conflict; we label the vertices in $A_i - C_i$ by 0, each edge with exactly one endpoint u in C_i by the same label as u and all remaining edges (which must have an endpoint in A_i) by 0. Note that the number of incident labels for each vertex in G_{i-1} does not change; and every vertex in $B_i - C_i$ becomes incident to at most $d_H := \kappa + 2$ different labels. Hence, this requirement of the theorem is fulfilled by induction.

It remains to show that any subgraph G'_i of G_i induced by (at most) $p-1$ labels has treewidth at most $c_H p$. Let $G'_{i-1} := G'_i[V(G_{i-1}) \cap V(G'_i)]$ and $\overline{B}'_i := G'_i[B_i]$. Let d be the omitted label and $D \subseteq C_i$ the set of vertices with label d from C_i; define $C' := C_i - D$ and $\overline{B}''_i := \overline{B}'_i - D$. Note that C' induces a clique in both G'_{i-1} and $\overline{B}''_i$ and that D has no neighbors in $B_i - C_i$ in $\overline{B}'_i$ because all edges incident to a vertex of D in $\overline{B}'_i$ have label d by our construction and are deleted. Hence, $G'_i = G'_{i-1} \oplus \overline{B}''_i$ and so its treewidth is bounded by the maximum of the treewidth of these two graphs. But by the induction hypothesis and Lemma 6.7, this number is bounded by $c_H p$. □

6.1.4 Approximation Algorithms and PTAS

We improve *all* the generic approximation and PTAS results given by Demaine et al. in [DHK05] (specifically, Theorems 3.3–3.7) and also by Grohe in [Gro03] by removing the dependence on $|H|$ from the exponent of n in the presented algorithms. This is due to Theorem 6.8 and also the fact that Lemma 6.6 corresponds to [Gro03, Lemma 16] as applied to tree decompositions that are *weakly* over $\mathcal{L}(\lambda, \mu)$. Nothing else in the proofs and algorithms needs to be changed. We refrain from re-stating all the generic results and highlight only some important concrete corollaries below.

Corollary 6.10. *There exists a 2-approximation algorithm for* Coloring *an H-minor-free graph in time $\mathcal{O}_H(n^{\mathcal{O}(1)})$.*

Corollary 6.11. *There exists a* PTAS *for* Independent Set, Minimum Color Sum, Vertex Cover, Max-Cut, *and* Maximum P-Matching *in H-minor-free graphs running in time $\mathcal{O}_{H,\varepsilon}(n^{\mathcal{O}(1)})$.*

The following result is of particular interest, as it does not follow from Theorem 6.8 but requires the techniques of [Gro03] using Lemma 6.6:[2]

Corollary 6.12. *There exists a* PTAS *for* Dominating Set *in H-minor-free graphs running in time $\mathcal{O}_{H,\varepsilon}(n^{\mathcal{O}(1)})$.*

For all the problems mentioned above, our method results in the first algorithm with this running time. The class of problems to which these techniques apply is very large and includes all the problems originally considered by Baker [Bak94] and also most minor-bidimensional problems, whereas for the latter, other known techniques also result in such PTASes [DH05a, FHL08].

[2] As this lemma is assumed by [DGK07], one can see this as a direct corollary of their work.

6.2 A Technique for (Nearly) Subexponential FPT-Algorithms

In this section, we introduce the technique of *guess and conquer* that for a wide range of problems shows their membership in SUBEPT$^+$. We first present the technique for the generic problem of finding a subgraph with a property; then we show how the idea can be used for domination, covering, and further types of problems. At the end of the section, we discuss the relation of the proposed algorithm to kernels.

6.2.1 The Technique of Guess and Conquer

We state our main technique for a broad class of parameterized problems. Given a *graph property* π, which is simply a set of directed or undirected graphs, we consider the following generic problem:

> k-Subgraph with Property π: Given a graph G, does G contain a subgraph with at most k vertices that has property π, i.e. is isomorphic to some graph in π?

The problem is abbreviated as k-sp (π). If we insist on finding *induced* subgraphs with property π, we use the notation k-isp (π) and if we want k to be the number of edges in an edge-induced subgraph then the problem is denoted by k-eisp (π). We allow that some vertices in the graphs in π have *fixed labels*, in which case, the task becomes to find a subgraph of a (partially) labeled graph G isomorphic to a graph in π, so that the labels match. Another variant is that we are additionally given a set $R \subseteq V(G)$ of *roots* (or terminals) in G and we are seeking a subgraph with property π that contains all the roots. We use the letters l and r to account for the labeled and rooted version of the problem, respectively, and the letter d to emphasize that we are dealing with directed graphs. Finally, we might be given a vertex- or edge-weighted graph and our goal is to find among all subgraphs of G with property π and at most k vertices (or edges), the one of *minimum* or *maximum* weight. We denote this whole class of problems by $(\{\text{Min,Max}\})k\text{-}\{\text{d,r,l,e,i}\}\text{sp}(\pi)$.

For example, k-Steiner Tree can be seen as a Min k-rsp(π) problem, where π is the set of all trees and R is the set of terminals that are to be connected in G. Likewise, one could look for a biconnected subgraph of size at most k containing a given set of terminals by taking π to be the set of all biconnected graphs. Also, Directed st-k-Path is an instance of k-dleisp (π) where π contains only a directed path of length k, in which the first vertex is labeled s and the last vertex is labeled t. Other interesting choices for π include being chordal, bipartite, edge-less (Independent Set), of maximum degree $r \geq 1$, a clique, planar, or containing only/avoiding cycles of specified length [DHK05]. We obtain the following general result:

Theorem 6.13. *Let π be a graph property such that on graphs of treewidth t one can find a (maximum/minimum weight/rooted/labeled/induced) subgraph with property π in time $\mathcal{O}(2^{\mathcal{O}(t)}n^{\mathcal{O}(1)})$. For any (directed/partially labeled) H-minor-free graph G, there exists an algorithm $\mathcal{A}$ solving problem $(\{\text{MIN,MAX}\})k$-$\{\text{D,R,L,E,I}\}\text{SP}(\pi)$ and that for any $\alpha \geq 1$ and fixed $\delta > 0$ runs in time $\mathcal{O}(2^{\mathcal{O}_H(\sqrt{k \log n})}n^{\mathcal{O}(1)}) = \mathcal{O}(2^{\mathcal{O}_H(k/\alpha)} + n^{\mathcal{O}(\alpha)})$ $= \inf_{0<\varepsilon\leq 1} \mathcal{O}((1+\varepsilon)^k + n^{\mathcal{O}_H(1/\varepsilon)}) = o(n^{\mathcal{O}(1)+\delta\sqrt{k}})$. In particular, the considered problem belongs to* SUBEPT$^+$.

Proof. Let p be some fixed integer; apply Theorem 6.8 to G to obtain a partition $V_1, \ldots, V_p$ of the vertex set of the graph, so that the graph induced by any $p-1$ of the sets has treewidth at most $c_H p$; such partition can be found in time $\mathcal{O}_H(n^{\mathcal{O}(1)})$. Now, consider an optimal subgraph $S^\star$ fulfilling the requirements of the problem; since $S^\star$ is assumed to have at most k vertices, there exists an $i^\star \in \{1, \ldots, p\}$, so that $V_{i^\star}$ contains at most $\lfloor k/p \rfloor$ vertices of $S^\star$. Since we do not know the value of $i^\star$, we simply *guess* it; there are at most p possibilities to do so and we try all of them. Hence, for each $i \in \{1, \ldots, p\}$, we repeat the following (see Figure 6.2 for an illustration):

For a fixed i, we have to determine which vertices of V_i belong to $S^\star$; once more, since we do not know these vertices, we simply *guess* them; there are at most $n^{\lfloor k/p \rfloor}$ possible subsets to try because we assumed that V_i contains at most $\lfloor k/p \rfloor$ vertices of $S^\star$. For each such subset $X \subseteq V_i$, we consider the subgraph $G' = G[(V(G) - V_i) \cup X]$. The treewidth of this subgraph is at most $c_H p + \lfloor k/p \rfloor$, and hence, we can find an optimal solution in G' in time $\mathcal{O}(2^{\mathcal{O}(c_H p + k/p)}n^{\mathcal{O}(1)})$ and we are done.

The algorithm's total running time is $\mathcal{O}(2^{\mathcal{O}(c_H p + k/p + k \log n/p)}pn^{\mathcal{O}(1)})$. This expression is minimized for $p = \left\lfloor \sqrt{k \log n / c_H} \right\rfloor$ resulting in a running time of $\mathcal{O}(2^{\mathcal{O}_H(\sqrt{k \log n})}n^{\mathcal{O}(1)})$. Since for any fixed $\delta' > 0$ we have that $2^{\sqrt{\log n}} = o(n^{\delta'})$, we can choose δ' in such a way that for any given fixed $\delta > 0$, the running time is $o(n^{\mathcal{O}(1)+\delta\sqrt{k}})$. On the other hand, for any $\alpha \geq 1$, if $c_H k \leq \alpha^2 \log n$, we have $\sqrt{c_H k \log n} + \log n \leq 2\alpha \log n$; and if $c_H k > \alpha^2 \log n$, we have $\sqrt{c_H k \log n} + \log n < 2c_H k/\alpha$. Hence, the running time is bounded by $\mathcal{O}(2^{\mathcal{O}_H(k/\alpha)} + n^{\mathcal{O}(\alpha)})$. By choosing $\alpha = \Theta(c_H/\ln(1+\varepsilon)) = \Theta(c_H/\varepsilon)$, we obtain a running time of $\inf_{0<\varepsilon\leq 1} \mathcal{O}((1+\varepsilon)^k + n^{\mathcal{O}_H(1/\varepsilon)})$. □

Using the result of Dorn et al. [DFT08] that the following problems are in EPT on (some) H-minor-free graphs when parameterized by treewidth, we immediately obtain:

Corollary 6.14. *For any graph H, the problem* Directed k-Path *is in* SUBEPT$^+$ *when restricted to H-minor-free graphs; the same is true for k-*Steiner Tree *at least on bounded-genus graphs.*[3]

The two problems mentioned above are prominent problems that were not known to admit FPT-algorithms with running time better than $\mathcal{O}(2^k n^{\mathcal{O}(1)})$ before, even on planar graphs. Besides improving on the best known FPT-algorithms for these problems, our result shows that it is very likely that they indeed admit subexponential FPT-algorithms.

[3] In [DFT08] it is claimed that Steiner Tree is in EPT on H-minor-free graphs when parameterized by treewidth; however, I know by private communication that at this time, a proof actually exists only up to bounded-genus graphs. The same is true for Connected Dominating Set.

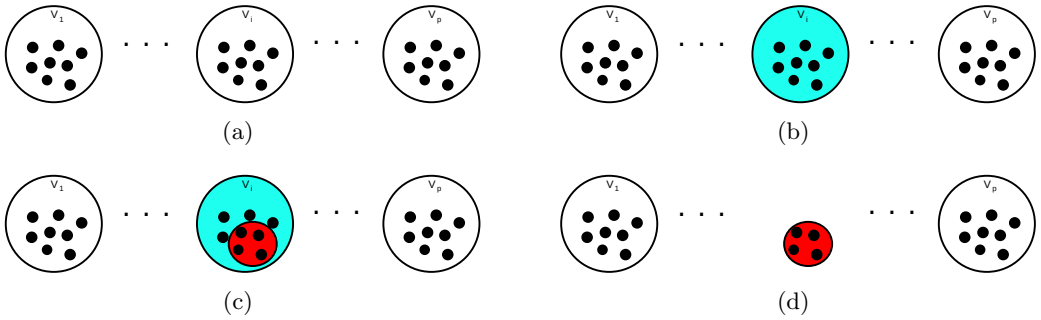


Figure 6.2: Illustration of Guess & Conquer: (a) partition the graph into p parts by Theorem 6.8; (b) *guess* which part contains a small part of the solution; (c) *guess* this small part of the solution; (d) *conquer* the remaining graph of small treewidth.

6.2.2 Guess and Conquer for Domination, Covering, and More

We introduced our technique for the class of k-{D,R,L,E,I}SP (π) problems, where we are looking for a subgraph with a certain property. Whereas many problems can be formulated as an instance of this generic problem class, some others like k-Vertex Cover, k-Dominating Set, or k-Leaf Tree and variants can not. We capture another class of problems by the following theorem.

Theorem 6.15. *Let Π be a problem that takes as input a graph G and outputs a set $S \subseteq V$ of vertices, and let k-Π its parameterization by $|S|$. Suppose that*

(i) on graphs of treewidth t, Π can be solved in time $\mathcal{O}(2^{\mathcal{O}(t)}n^{\mathcal{O}(1)})$; and

(ii) if for an edge $e \in E(G)$ it is known that some solution of S excludes both endpoints of e then Π can be reduced to finding a solution in $G - e$; that is, there exists a $k' \leq k$ such that given a solution for $(G - e, k')$, one can compute a solution for (G, k) in polynomial time.

Then for any graph G excluding a fixed minor H, there exists an algorithm $\mathcal{A}$ solving k-Π on instance (G, k) such that for any $\alpha \geq 1$ and fixed $\delta > 0$, algorithm $\mathcal{A}$ runs in time $\mathcal{O}(2^{\mathcal{O}_H(\sqrt{k \log n})}n^{\mathcal{O}(1)}) = \mathcal{O}(2^{\mathcal{O}_H(k/\alpha)} + n^{\mathcal{O}(\alpha)}) = \inf_{0<\varepsilon\leq 1} \mathcal{O}((1+\varepsilon)^k + n^{\mathcal{O}_H(1/\varepsilon)}) = o(n^{\mathcal{O}(1)+\delta\sqrt{k}})$ In particular, k-Π belongs to SUBEPT$^+$.

Proof. Let p be a fixed integer and let $E_1, \ldots, E_p$ be the edge partition obtained by Theorem 6.9, so that the graph induced by any $p-1$ of the parts has treewidth at most $c_H p$ and furthermore, each vertex appears in at most d_H of the parts. Let $S^\star$ be an optimal solution to $\mathcal{P}$ having at most k vertices; then the total number of appearances of vertices in $S^\star$ in the parts $E_1, \ldots, E_p$ is bounded by $d_H k$, where d_H is the constant from Theorem 6.9. It follows that there exists an $i^\star \in \{1, \ldots, p\}$ such that the graph induced by $E_{i^\star}$ contains at most $\lfloor d_H k/p \rfloor$ vertices of $S^\star$. We *guess* the value of $i^\star$ and the set of vertices $X^\star := S^\star \cap E_{i^\star}$ by trying all $pn^{\lfloor d_H k/p \rfloor}$ possibilities.

Because of assumption (ii), we can delete all the edges in $E_{i^\star}$ that do not have an endpoint in $X^\star$. The graph $G - E_{i^\star}$ has treewidth at most $c_H p$ and by adding the vertices of $X^\star$ to every bag in such a tree decomposition, the width becomes at most $c_H p + \lfloor d_H k/p \rfloor$. By choosing $p := \lfloor \sqrt{k \log n} \rfloor$ and repeating the analysis in the proof of Theorem 6.13, we obtain our result. □

The k-Vertex Cover problem satisfies property (ii) above because if for an edge e, we know that both endpoints do not belong to the solution, then we can reject, since e is not covered. For k-Dominating Set, such an edge is simply irrelevant, even for the connected version. That Connected k-Dominating Set fulfills property (i) was shown by Dorn et al. [DFT08] (see footnote on page 120). Hence, we have

Corollary 6.16. ({Connected, Independent}) k-Dominating Set *and* ({Connected, Independent}) k-Vertex Cover *(at least) in bounded-genus graphs belong to the class* SUBEPT$^+$.

Still, Theorems 6.13 and 6.15 do not capture all problems to which the basic idea of our technique applies; for example, a modification of the proof of Theorem 6.15 shows that the technique also works for the undirected k-Leaf Tree problem. But since this problem is known to be in SUBEPT by the theory of bidimensionality, we refrain from presenting the details. It would be interesting to see if (a modification of) our technique can be used to solve the directed version of this problem.

Another interesting problem is k-Bounded Degree Deletion(d), or k-BDD(d) for short, where we want to delete a set of at most k vertices so that the remaining graph has degree at most d. Note that k-BDD(0) is equivalent to k-Vertex Cover. Whereas we can not ignore edges that are known not to have endpoints in the solution, we can delete such edges and store at each vertex, the maximum allowed degree that remains; this information can then be incorporated in the dynamic programming on the bounded treewidth graph. The problem has a kernel with a linear number of vertices and is thus in SUBEPT on H-minor-free graphs but for the case where we seek a connected solution, we obtain that k-BDD(d) belongs to SUBEPT$^+$ only by applying our technique.

6.2.3 Further Analysis and Relation to Kernels

The analysis in the proof of Theorem 6.13 reveals that if $k = \mathcal{O}(\log n)$, then our algorithm runs in polynomial time; on the other hand, if $k = \omega(\log n)$, i.e. if k is known to be at least $\Omega(\iota(n) \log n)$ for any computable, non-decreasing and unbounded function $\iota : \mathbb{N} \to \mathbb{N}$, then we have a SUBEPT algorithm with time complexity $2^{\mathcal{O}_H(k/\sqrt{\iota(k)})}$ (see proof of Theorem 6.13 with $\alpha = \sqrt{\iota(n)}$). But the condition $k = \omega(\log n)$ is nothing else but asking for a *subexponential kernel*; consider the following definition that we use to state the subsequent corollary (note that a polynomial kernel implies $\log n \leq c \log k$).

Definition 6.17. We say a parameterized problem k-Π admits a *minor-monotone subexponential kernel* if it can be reduced in polynomial time to an equivalent instance of size at most $2^{\mathcal{O}(k/\iota(k))}$ via edge contractions and deletions for some computable, non-decreasing, and unbounded function $\iota : \mathbb{N} \to \mathbb{N}$.

Corollary 6.18. *Let k-Π be a parameterized problem on H-minor-free graphs that can be solved in time $\mathcal{O}(2^{\mathcal{O}_H(\sqrt{k \log n})} n^{\mathcal{O}(1)})$ and admits a minor-monotone subexponential kernel. Then k-Π belongs to* SUBEPT*. In particular, if k-Π admits a minor-monotone polynomial kernel then it can be solved in* SUBEPT*-time $\mathcal{O}(2^{\mathcal{O}_H(\sqrt{k \log k})} n^{\mathcal{O}(1)})$.*

Any parameterized problem that can be solved in time $\mathcal{O}(f(k)n^{\mathcal{O}(1)})$ admits a kernel of size $f(k)$ [Nie02]. It follows that all problems in SUBEPT also have a subexponential kernel. Our corollary above shows the reverse direction of this observation for the problems that admit our technique on H-minor-free graphs; for these problems, we obtain that a subexponential FPT algorithm exists if and only if a minor-monotone subexponential kernel can be constructed.

Note that in the statement of Theorems 6.13 and 6.15, the parameter α is not required to be fixed; it can be any non-decreasing function from $\mathbb{N}$ to $\mathbb{N}$. Hence, the running time of the algorithm $\mathcal{A}$ that is obtained by these theorems cannot only be bounded subexponentially in k but instead, slightly super-polynomially in n. For example, by choosing $\alpha = \log \log n$, we obtain a bound of $\mathcal{O}(2^{\mathcal{O}(k/\log \log k)} + n^{\mathcal{O}(\log \log n)})$.

6.3 Algorithms on Odd-Minor-Free Graphs

In [DHK10], Demaine et al. prove a structural decomposition theorem for odd-minor-free graphs that is very similar to the RS-decomposition theorem for H-minor-free graphs [RS03]. They also present an algorithm running in time $n^{\mathcal{O}_H(1)}$ to compute such a decomposition. However, upon inspecting their proof, we obtain the following simpler intermediate result that turns out to be more useful for algorithmic purposes when combined with known results on H-minor-free graphs; in particular, it can be used to obtain FPT-versions of various algorithms when combined with our results from Section 6.1. Let $\mathscr{B}(\mu)$ denote the class of all bipartite graphs augmented by at most μ additional vertices called apices. We have

Theorem 6.19 (adapted from [DHK10])**.** *Let G be a given odd-H-minor-free graph. There exists a fixed graph H' depending only on H and an explicit uniform algorithm that computes a tree decomposition with adhesion at most κ of G that is strongly over the union of $\mathscr{B}(\mu)$ and the class of H'-minor-free graphs, where μ and κ are computable functions depending only on H. Furthermore, we have the following properties:*

- *(i) the H'-minor-free graphs appear only in the leaves of the tree decomposition;*
- *(ii) if B_2 is a bag that is a child of the bag B_1 in the tree decomposition and $\overline{B}_1$ consists of a bipartite graph W together with at most μ apices then $|B_2 \cap V(W)| \leq 1$;*
- *(iii) the at most μ apices of each bag are also computed;*
- *(iv) the algorithm runs in time $\mathcal{O}_H(n^4)$.*

Proof. The decomposition algorithm of Demaine et al. [DHK10, Theorem 4.1] basically works as follows: if the given graph contains a certain fixed bipartite graph H' as a minor, find a bipartite graph with some apices, create one bag out of it, and iterate this process on the components of the remaining graph; otherwise the graph excludes

H' as a minor and the RS-decomposition can be applied. Now instead of applying the RS-decomposition at this step (which is Step (7) of the algorithm in [DHK10]), we simply create a bag containing the current subgraph and connect it as a leaf to the tree decomposition. All other properties follow directly from [DHK10, Theorem 4.1] and its proof and analysis. □

Together with Theorem 6.5 we obtain the following corollary:

Corollary 6.20. *There is an explicit uniform algorithm that, given an odd-H-minor-free graph G, computes a tree decomposition $(T, \mathcal{B})$ of G with adhesion at most κ in time $\mathcal{O}_H(n^{\mathcal{O}(1)})$ such that for every bag $B \in \mathcal{B}$, we have either*

(i) the companion of B is in $\mathcal{L}(\lambda, \mu)$; or
(ii) the closure of B is in $\mathscr{B}(\mu)$,

where λ, μ, and κ are computable functions depending only on $|H|$. The μ apices of (the companion of) each bag in $\mathcal{B}$ can be computed in the same time bound. Moreover, if $B_1, B_2 \in \mathcal{B}$ and B_2 is a child of B_1 in the tree decomposition and $\overline{B}_1$ consists of a bipartite graph W together with at most μ apices then $|B_2 \cap V(W)| \leq 1$.

Proof. We first apply Theorem 6.19 to obtain a tree decomposition $(T, \mathcal{B})$ as described. Then we consider each leaf that contains an H'-minor-free subgraph G' of G and apply Theorem 6.5 to it to obtain a tree decomposition $(T_{G'}, \mathcal{B}_{G'})$ that is weakly over $\mathcal{L}(\lambda, \mu')$; afterwards, we add the intersection of $V(G)'$ with its parent bag in $(T, \mathcal{B})$ to each bag of $\mathcal{B}_{G'}$ and replace the leaf of $(T, \mathcal{B})$ containing G' with this finer tree decomposition. This way, we make sure that the number of apices in every bag of the global tree decomposition is the maximum of the value obtained from Theorem 6.19 and $\mu' + \kappa$ and let μ be this maximum. □

The proof of the following theorem is analogous to the proof of Theorem 6.9; only note that the vertex set of a graph from $\mathscr{B}(\mu)$ naturally has a partition into 2 parts of bounded treewidth: just take each part of the bipartition together with some apices (see also the proof of [DHK10, Theorem 1.1]).

Theorem 6.21. *For any fixed graph H there is a constant c_H such that for every odd-H-minor-free graph G, the vertices of G can be partitioned into 2 parts such that each of the parts induces a graph of treewidth at most c_H. Furthermore, such a partition can be found in explicit uniform FPT-time, i.e. $\mathcal{O}_H(n^{\mathcal{O}(1)})$.*

This is the best possible analog to the Baker-style decomposition of Theorem 6.8 for odd-minor-free graphs since these graph classes include all bipartite graphs; and complete bipartite graphs can not be partitioned into more than 2 parts of bounded treewidth. A direct corollary is the following:

Corollary 6.22. *There exists a 2-approximation algorithm for* Coloring *an odd-H-minor-free graph in time $\mathcal{O}_H(n^{\mathcal{O}(1)})$.*

Also, 2-approximations with the same FPT-running time for various other problems, such as many of the ones mentioned in Section 6.1.4, can be obtained. See [DHK05] and [DHK10] for more details.

6.3.1 PTASes on Odd-Minor-Free Graphs

Grohe [Gro03] showed that various problems admit a PTAS on H-minor-free graphs. Most of these PTASes *can not* be generalized to odd-minor-free graphs as they would imply corresponding PTASes on bipartite or even general graphs for APX-hard problems. However, Demaine et al. ask several times in [DHK10] whether the PTASes for Vertex Cover and Independent Set can be generalized to odd-minor-free graphs; this seems plausible since these two problems can be solved in polynomial time on bipartite graphs. Indeed, we are able to answer this question affirmatively in this section. To this end, we define the *take-or-leave* version of these problems as follows: every vertex of the graph is associated with two numbers w^+ and w^-; if a vertex is chosen to be in the solution, i.e. in the vertex cover or independent set, it contributes a value of w^+ to the objective function; if it is not included in the solution, it contributes w^- to the objective function (the usual unweighted variants are then special cases of the take-or-leave version where $w^+ = 1$ and $w^- = 0$ for every vertex).

Lemma 6.23. *The take-or-leave versions of* Vertex Cover *and* Independent Set *can be solved in polynomial time on bipartite graphs.*

Proof. Just note that the matrices used in the standard linear programming formulations of the unweighted version of these problems are totally unimodular for bipartite graphs, and hence all the corners of the corresponding polyhedra are integral. But the only thing that changes now is the objective function; in particular, the polyhedron is still the same and integral. Hence, we can find a solution in polynomial time just by solving these linear programs. □

Theorem 6.24. *There exists a* PTAS *for* Vertex Cover *and* Independent Set *in odd-H-minor-free graphs running in time $\mathcal{O}_{H,\varepsilon}(n^{\mathcal{O}(1)})$.*

Proof. We prove the theorem for Vertex Cover; the case of Independent Set is analogous. Let G be a given odd-H-minor-free graph. We first compute a tree decomposition $(T, \mathcal{B})$ of G as specified by Theorem 6.19 and root it at some bag containing a graph from $\mathscr{B}(\mu)$; if such a bag does not exist, the graph is actually H'-minor-free and we obtain our result by Corollary 6.11. For every bag $B \in \mathcal{B}$ we define the *subproblem at B* to be the considered problem on the subgraph of G that is induced by B and all of its descendants in T. We perform dynamic programming from the leaves of the tree decomposition to the root and store at each bag B the following information: for every subset U of the apices A of B, we compute a solution for the subproblem at B that must contain U, must not contain $A - U$, and is within a factor of $(1+\varepsilon)$ of the optimal solution with these properties; we let $\nu(B, U)$ be the value of such a solution minus $|U|$ and store it in a table; if a solution with these properties does not exist, we store $\perp$ to denote this fact. Here we assume w.l.o.g. that we have nice apex sets that include the intersection of the current bag with its parent bag; in fact, for leaves of the tree that contain an H'-minor-free graph, the apex set is defined to be this intersection. For such leaves, we can compute the values of the dynamic programming table by invoking 2^κ times the PTAS from Corollary 6.11 – once for each subset $U \subseteq A$.

Now suppose we are at a bag B with children $B_1, \ldots, B_t$ ($t \geq 0$) and B contains a graph from $\mathscr{B}(\mu)$. Suppose B contains a bipartite graph W together with apices A. For each fixed selection $U \subseteq A$ of the apices, we have to compute a near optimal solution $\nu(B, U)$. For each child B_i, define $Y_i = B_i \cap B$ and for each set $X \subseteq Y_i$, let $\nu^\star(B_i, X)$ be the value of a $(1+\varepsilon)$-approximate solution for the subproblem at B_i that contains X but not $Y_i - X$, minus $|X|$ (or $\perp$ if such a solution does not exist). Note that the value of $\nu^\star(B_i, X)$ can be looked up in the dynamic programming table of B_i by taking the minimum over all $\nu(B_i, U') + |U'| - |X|$ with $U' \cap Y_i = X$. These values can be precomputed and stored.

Note that the status of every vertex in Y_i, i.e. whether or not it should be in the solution, is completely specified by the choice of U except for at most one vertex $v \in Y_i \cap V(W)$. For each vertex $v \in V(W)$, define its *taking* weight w^+ as $1 + \sum \nu^\star(B_i, \{v\} \cup (U \cap Y_i))$ and define its *leaving* weight w^- as $\sum \nu^\star(B_i, U \cap Y_i)$ where the sums go over all children B_i that contain v. We solve the take-or-leave version of the problem with these weights on W using Lemma 6.23 and store it as the solution for $\nu(B, U)$ (or $\perp$ if no such solution exists). We return the solution of minimum value stored at the root of the tree decomposition. This finishes the description of the algorithm.

The correctness is immediate at the leaves of the dynamic program. For a non-leaf bag $B \in \mathcal{B}$ with bipartite graph W and apices A and a given selection $U \subseteq A$ of its apices, let $S^\star$ be an optimal solution for the subproblem at B that includes U but not $A - U$, and S be the solution corresponding to $\nu(B, U)$ as computed by our algorithm. For a set $X \subseteq V(W)$, let OPT_X denote the optimal solution value for the subproblem at B given that $U \cup X$ must be in the solution and $B - U - X$ must not be in the solution; let TOL_X denote the objective function value of the take-or-leave problem defined at B if X is taken as the solution, plus $|U|$. By our construction and the induction hypothesis of the dynamic program, for any set $X \subseteq V(W)$, we have $\mathrm{TOL}_X \leq (1+\varepsilon)\,\mathrm{OPT}_X$. Hence, we obtain $|S| = \mathrm{TOL}_{S \cap V(W)} \leq \mathrm{TOL}_{S^\star \cap V(W)} \leq (1+\varepsilon)\,\mathrm{OPT}_{S^\star \cap V(W)} = (1+\varepsilon)|S^\star|$. □

Note that Theorem 6.24 holds also for the vertex-weighted versions of these problems; the proof is analogous.

6.3.2 Subexponential FPT for Odd-Minor-Free Graphs

Another question that is asked by Demaine et al. [DHK10] is whether k-Vertex Cover and k-Independent Set admit SUBEPT-algorithms on odd-minor-free graphs. As in the case of the PTASes, these are basically the only problems for which this seems possible as such algorithms for most other prominent problems would contradict hardness results in parameterized complexity. Indeed, we can obtain subexponential parameterized algorithms for these problems in a similar way as the PTASes above. First, let us state the following known result.[4]

Lemma 6.25 (partly taken from [DH05b, DFHT05])**.** *There exists an algorithm that, given an H-minor-free graph G and an integer k, runs in time $\mathcal{O}(2^{\mathcal{O}_H(\sqrt{k})} n^{\mathcal{O}(1)})$ and*

[4] I would like to thank Fedor Fomin for a helpful discussion on this matter.

(i) decides if G contains a vertex cover of size at most k and in this case, returns a minimum vertex cover of G; and
(ii) decides if G contains an independent set of size at least k and if this is not *the case, returns an independent set of maximum size in G.*

Proof. The algorithm for k-Vertex Cover follows directly from the bidimensionality theory [DH05b, DFHT05].

As for k-Independent Set, it is a well-known fact that H-minor-free graphs have bounded average degree [Tho01] and hence, an independent set of size $\Omega_H(n)$. So, all we have to do is to count the number of vertices of G; if this is at least $c_H k$, for a suitable constant c_H, the answer is "yes". Otherwise, the size of the graph is bounded by $c_H k$ and hence has treewidth at most $\mathcal{O}_H(\sqrt{k})$ [AST90]; an optimal independent set can be found by standard dynamic programming. □

Theorem 6.26. *There exists an algorithm that, given an odd-H-minor-free graph G and an integer k, runs in time $\mathcal{O}(2^{\mathcal{O}_H(\sqrt{k})}n^{\mathcal{O}(1)})$ and*

(i) decides if G contains a vertex cover of size at most k and in this case, returns a minimum vertex cover of G; and
(ii) decides if G contains an independent set of size at least k and if this is not *the case, returns an independent set of maximum size in G.*

Proof. The proof is basically analogous to the proof of Theorem 6.24 above. We start by computing a tree decomposition as given by Theorem 6.19. For each bag B of and each selection $U \subseteq A$ of the apices of the bag, we compute a value $\nu(B, U)$, specifying if the subproblem at B contains a vertex cover of size at most k that includes U but not $A - U$ and if so, the size of the minimum vertex cover with these properties; if not, we store $\perp$ to indicate a negative answer. If all entries for a bag B turn out to be $\perp$, the answer to problem is "no" and we can terminate.

The entries for the leaves that contain an H'-minor-free graph can be computed using Lemma 6.25. For a (leaf or non-leaf) bag B that contains a graph from $\mathscr{B}(\mu)$ we construct a take-or-leave version of k-Vertex Cover analogously to what we did in the proof of Theorem 6.24 and solve it in polynomial time. The correctness follows as in Theorem 6.24 and the running time is dominated by the running time of the algorithm on the H'-minor-free leaves and hence is subexponential FPT in k.

The case for k-Independent Set is analogous only that if at any point the answer to some subproblem is "yes" we may return this answer and terminate; otherwise, we proceed as above. □

6.4 Conclusion and Outlook

We significantly accelerated one of the main tools in PTAS design – namely, Baker's decomposition – on all proper minor-closed graph classes, thereby obtaining the improvement for all the applications of this technique. We showed similar results for odd-minor-free graphs and obtained the first PTAS and subexponential FPT-algorithms for

k-Vertex Cover and k-Independent Set on these graph classes. Based on Baker's approach, we also introduced the technique of Guess & Conquer for designing (nearly) subexponential FPT-algorithms on these graph classes. We improved the best known FPT-algorithms for k-Steiner Tree and Directed k-Path in H-minor-free graphs that were previously only known to be in EPT even on planar graphs (in the case of Steiner Tree). We actually conjecture that these problems are in SUBEPT on H-minor-free graphs and repeat this as an important open question for future work.

A further important question in this area is whether k-Subgraph Isomorphism, parameterized by the size of the subgraph pattern, admits a SUBEPT or at least SUBEPT$^+$ algorithm on H-minor-free or at least planar graphs. The main difficulty is that even if the host graph has small treewidth, current algorithms still need linear exponential time in k [Dor10] to decide this problem. But in order to apply our technique, we would need an algorithm that is in EPT with respect to treewidth but polynomial in k. See [Dor10] for recent progress on this problem.

Part III

Lower Bounds for the Complexity of Monadic Second-Order Logic

7 On Brambles, Grid-Like Minors, and Tree-Ordered Webs[1]

In this chapter, we develop several algorithmic tools – on one hand, for a number of (well-known) concepts from structural graph theory, and on the other hand, for some new structures that we introduce in our work. While these algorithms stand in their own right and are of independent interest, we will exploit them, in particular, to prove lower bounds for the tractability of monadic second-order logic (MSO_2) in the next chapter. .

The concept of treewidth has received immense attention ever since it was introduced, especially because many NP-hard problems can be handled efficiently on graphs of bounded treewidth (e.g. all problems that can be defined in MSO_2 [Cou90]). The dual notion to treewidth is the concept of a *bramble* [ST93, Ree97]; a bramble of large order is a witness for large treewidth.

Definition 7.1 (Bramble)**.** Let G be a graph. Two subgraphs B, B' of G *touch* if they share a vertex or if there is an edge $e \in E(G)$ joining B and B'. A *bramble* in G is a set $\mathscr{B}$ of connected subgraphs of $V(G)$ such that any two $B, B' \in \mathscr{B}$ touch. The subgraphs in $\mathscr{B}$ are called *bramble elements*. A set $S \subseteq V(G)$ is a *hitting set* for $\mathscr{B}$ if it intersects every element of $\mathscr{B}$. The order of $\mathscr{B}$ is the minimum size of a hitting set.

A canonical example of a bramble is the set of crosses (union of a row and a column) of an $\ell \times \ell$-*grid* (see Figure 7.1). The following theorem shows the duality of treewidth and brambles:

Theorem 7.2 (Seymour and Thomas [ST93])**.** *A graph G has treewidth at least ℓ if and only if G contains a bramble of order at least $\ell + 1$.*

It turns out that so far, brambles have received far less attention than tree decompositions; perhaps the reason is that brambles can look quite complex and do not necessarily have a "nice" structure to be dealt with reasonably. Indeed, Robertson and Seymour figured out that there are certain brambles with "very nice" structure that are much more useful than general brambles: namely, a *grid-minor of large order*. In fact, Robertson and Seymour show that a class of graphs has bounded treewidth if and only if it excludes a fixed grid as a minor [RS86]. A grid is a canonical planar graph and the existence of large grids has various algorithmic and non-algorithmic applications and implications, e.g. [RS95, Epp00a, Gro04, DFHT05, Gro07b, CSH08, Kre09b]. However, the best known bounds relating treewidth and grid-minors are the following:

[1] This chapter is based on joint work with Stephan Kreutzer [KT10a, KT10b].

(a) (b)

Figure 7.1: (a) An example of a bramble of order 2 with 3 elements indicated with the colors green, blue, and yellow; note that the lowest vertex is both blue and yellow; (b) the crosses of a grid are a canonical example of a bramble.

Theorem 7.3 (Robertson, Seymour, and Thomas [RST94])**.** *Every graph with treewidth at least* $20^{2\ell^5}$ *contains an* $\ell \times \ell$*-grid as a minor. There are graphs of treewidth* $\ell^2 \log \ell$ *that do not contain an* $\ell \times \ell$*-grid as a minor.*

So, there is a huge gap between the known lower and upper bounds of this theorem; Robertson and Seymour conjecture that the true value should be closer to the lower bound, i.e. that every graph should have a grid of order polynomial in the treewidth. Recently, Reed and Wood [RW08] attacked this problem by loosening the requirement for the bramble to be a grid; instead, they define a structure that they call a *grid-like minor*, as a replacement structure for a grid-minor, and prove that every graph does indeed contain a grid-like minor of order polynomial in the treewidth.

All of the results regarding brambles, grid-minors, and grid-like minors mentioned above are *existential*; to the best of our knowledge, it is not known so far how to *efficiently* construct *any* bramble of large order even when a tree decomposition of optimal width is given. It was not even studied up until recently, how large a bramble of the order of the treewidth can be; Grohe and Marx [GM09] showed that there exist brambles of size polynomial in the size of the graph whose order is roughly the square root of the treewidth (up to log-factors); but they also show that there exist graphs in which any bramble of order larger than the square root of the treewidth has size *exponential* in the size of the graph.

Constructing Brambles We provide the first polynomial-time algorithm to compute a bramble that is guaranteed to have the order of the square-root of the treewidth, up to log-factors, hence almost matching the best possible theoretical bound for polynomial-sized brambles. Our approach is based on the proof given in [GM09] but additionally, involves the approximation algorithms for treewidth, balanced separators, and sparse separators, which in turn are based on linear and semi-definite programming methods to obtain low-distortion metric embeddings of graphs [LR88, BGHK95, FHL08]. Even though we do not need to get into all of these topics in this work, it is interesting to note that it is a combination of all of these that finally gives rise to our algorithm. We also obtain an alternative (simpler) algorithm to construct a bramble of smaller size but lower order; in order to do so, we introduce the notion of a *k-web*, a structure that is similar to what Diestel et al. [DGJT99] denote by a *k-mesh*, and show that it can be

computed by a polynomial time algorithm.

Recently, Chapelle et al. [CMT09] presented an algorithm that computes a bramble of the order of the treewidth in time $\mathcal{O}(n^{k+4})$, where n is the size of the graph and k the treewidth; hence, they obtain brambles of optimal order but naturally, they need exponential time in order to do so. We would also like to mention a result by Bodlaender et al. [BGK05] who provide a polynomial-time *heuristic* to compute brambles in graphs; they use their algorithm for some computational experiments but do not prove any bounds on the order of the bramble they obtain.

Constructing Grid-Like Minors Afterwards, we turn our attention to grid-like minors and present the first polynomial-time algorithm to construct a grid-like minor of large order in general graphs. Again, our method is based on the original existence proof of [RW08] but involves a number of new ideas and techniques, most notably the following: first, we make use of k-webs instead of brambles, and second, we apply the very recent result of Moser and Tardos [Mos09, MT09] that provides an algorithmic version of the Lovász Local Lemma. These two ideas make it possible that the algorithmic bound that we obtain (i.e. the order of the grid-like minor we construct) is very close to the existential bound proved by Reed and Wood; if we "just" used our bramble algorithm and proceeded as in the original proof, the exponents would roughly have tripled.

Perfect Brambles and a Meta-Theorem As a first application of our results, we define the notion of a *perfect bramble* as a perhaps somewhat more "handy" replacement for grid-minors. Most notably, a perfect bramble defines a subgraph that has *bounded degree*, *large treewidth*, and has the property that every vertex appears in *at most* 2 bramble elements. We show that every graph contains a perfect bramble of order polynomial in the treewidth and that such a bramble can be computed in polynomial time. This shows that if the upper bound in Theorem 7.3 is to be improved to a polynomial, it is sufficient to prove it for perfect brambles.

Moreover, we present a *meta theorem* on perfect brambles: we show that essentially any graph parameter that is *subgraph monotone* and is *large on a perfect bramble* can be decided in time $\mathcal{O}(2^{\text{poly}(k)}\text{poly}(n))$ and that a *witness* can be provided in the same time bound; here n is the size of the input and k is the size of the parameter. Hence, we show that such parameters are in the class EXPT of singly exponential FPT-parameters.

One of the most important consequences of the graph minor theorem of Robertson and Seymour [RS95, RS04, FL88] is the following: for a given graph G and parameter $\pi(G)$ that is *minor monotone*, one can decide if $\pi(G) \leq k$, in $\mathcal{O}(f(k)n^3)$-FPT time for some function f. This is a very general and powerful theorem but there is a price to be paid: (i) for any such parameter, an algorithm is known to exist, but the algorithm itself can not be known in general; (ii) the theorem gives a *non-uniform* algorithm, meaning there is a different algorithm for every value of k; (iii) the function $f(k)$ is, in general, not computable and can be arbitrarily large. Frick and Grohe [FG01] proved explicit bounds for certain graph classes and parameters definable in first-order logic, though the bounds were still non-elementary. Demaine and Hajiaghayi [DH07] proved a

bound of $\mathcal{O}(2^{2^{\text{poly}(k)}}\text{poly}(n))$ for general graphs, when the considered parameter fulfills a few additional constraints. They use the grid-minor theorem together with ideas from the bidimensionality theory [DFHT05] to obtain this bound. By using a perfect bramble instead of a grid-minor, we can improve this bound to be singly exponential in k, although the constraints that we require are somewhat stronger than [DH07]; still, our technique can be applied to many problems to which their technique also applies.

Tree-Ordered Webs A k-web essentially gives us a large grid-like minor attached to a certain subcubic tree. What we aim at is to find a subgraph of a k-web that still contains a large grid-like minor attached to a tree but with the property that there exist MSO_2 formulas that distinguish the tree from the grid-like minor and furthermore, define a *linear order* on the vertices of the tree. This order induces an order on some parts of the grid-like minor and provides us with the notion of a *first row* with a unique linear order on a number of its first columns. This will enable us to interpret this structure as a certain colored wall and becomes the heart of the main proof of our result on the complexity of MSO_2 in subgraph-closed graph classes in the next chapter.

7.1 Constructing Brambles and k-Webs

In this section, we show two different methods to construct a bramble in a graph. The first one is based on a randomized construction by Grohe and Marx [GM09]; our second construction uses a *k-web*, a concept that we also introduce in this section, and that results in a bramble whose size does not depend on n.

The following theorem essentially says that if we are looking for a polynomial-sized bramble, the best order we can hope for is about the square-root of the treewidth:

Theorem 7.4 (Grohe and Marx [GM09])**.**

(i) Every n-vertex graph G of treewidth k has a bramble of order $\Omega(\frac{\sqrt{k}}{\log^2 k})$ and size $\mathcal{O}(k^{\frac{3}{2}} \cdot \ln n)$.

(ii) There is a family $(G_k)_{k\geq 1}$ of graphs such that:

- *$|V(G_k)| = \mathcal{O}(k)$ and $E(G_k) = \mathcal{O}(k)$ for every $k \geq 1$;*
- *$\text{tw}(G_k) \geq k$ for every $k \geq 1$;*
- *for every $\varepsilon > 0$ and $k \geq 1$, every bramble of G_k of order at least $k^{\frac{1}{2}+\varepsilon}$ has size at least $2^{\Omega(k^{\varepsilon})}$.*

The proof of Theorem 7.4 (i), as given in [GM09], proceeds as follows: first, it is shown that a large set exists that *does not admit sparse separators*; afterwards, a *maximum concurrent flow* is computed on this set; and finally, a randomized algorithm is presented that constructs a bramble based on the solution of the flow problem. It turns out the only step that is not (polynomially) algorithmic is the first step; we show how to obtain a suitable set of vertices efficiently below.

7.1.1 Finding A Large Set Lacking Sparse Separators

Recall that a *separator* of a graph G is a partition of its vertices into three classes (A, B, S) such that there are no edges between A and B. We allow A or B to be empty but require $S \neq \varnothing$. The *size* of a separator is the size of the set S (see Figure 5.1 (a)). For a subset $W \subseteq V(G)$, we say that a separator is γ-*balanced* or just a γ-*separator* with respect to W, if for every connected component C of $G - S$, we have $|W \cap C| \leq \gamma|W|$. The treewidth of a graph is closely related to the existence of balanced separators:

Lemma 7.5 (Reed [Ree92, Ree97])**.**

(i) If G has treewidth greater than $3k$, then there is a set $W \subseteq V(G)$ of size exactly $2k+1$ having no $\frac{1}{2}$-balanced separator of size k;

(ii) if G has treewidth at most k, then every $W \subseteq V(G)$ has a $\frac{1}{2}$-balanced separator of size $k+1$.

The *sparsity* of a separator (A, B, S) with respect to W is defined as

$$\alpha^W(A, B, S) = \frac{|S|}{|(A \cup S) \cap W| \cdot |(B \cup S) \cap W|}\,.$$

We denote by $\alpha^W(G)$ the minimum of $\alpha^W(A, B, S)$ over all separators (A, B, S). It is easy to see that for every connected G and nonempty W, $\frac{1}{|W|^2} \leq \alpha^W(G) \leq \frac{1}{|W|}$. We are interested in a set W with *no sparse separator*, i.e. where the sparsity of the sparsest vertex cut is close to the maximum. Grohe and Marx [GM09] showed that the non-existence of balanced separators can guarantee the existence of such a set W:

Lemma 7.6 (Grohe and Marx [GM09])**.** *If $|W| = 2k+1$ and W has no $\frac{1}{2}$-balanced separator of size k in a graph G, then $\alpha^W(G) \geq \frac{1}{4k+1}$.*

The proof of Lemma 7.5 is algorithmic, but the algorithm is not polynomial-time since deciding if a (set in a) graph has a balanced separator of size k is an NP-complete problem. Hence, we have to work with approximations. On the other hand, Grohe and Marx note that Lemma 7.6 does not remain true for larger W by showing an example with $|W| = 4k$ and $\alpha^W(G) = \mathcal{O}(1/k^2)$; so, if we work with approximations, we can not use this lemma directly. We show in this section how to circumvent these problems by presenting a polynomial-time algorithm to find a large set W with no sparse separator. Our algorithm follows the framework of approximating balanced separators by using sparse separators, as introduced by Leighton and Rao [LR88]. Additionally, we make use of the following two results:

Lemma 7.7 (Feige et al. [FHL08])**.** *Let G be a connected graph, $W \subseteq V(G)$, and T be the optimal $\frac{2}{3}$-separator of W in G. There exists a polynomial-time algorithm that computes a separator (A, B, S) of G such that $\alpha^W(A, B, S) \leq \beta_0 \alpha^W(G)\sqrt{\log |T|}$, for some constant β_0.*

Lemma 7.8 (adapted from Bodlaender et al. [BGHK95])**.** *Let G be a graph and $s \in \mathbb{N}$ be given. Suppose that for any connected subset U of $V(G)$ and given set $W \subseteq U$ with $|W| = 4s$, there exists a $\frac{3}{4}$-separator of W in U of size at most s and that such a separator can be found in polynomial time. Then the treewidth of G is at most $5s$ and an according tree decomposition can be found in polynomial time.*

Corollary 7.9. *Given a graph G of treewidth $k^\star$, there is a polynomial-time algorithm that constructs a tree decomposition of width k_1, such that for constants c_0, c_1, c_2, we have*

(i) $\frac{k_1}{c_0\sqrt{\log k_1}} \leq k^\star \leq k_1 \leq c_0 k^\star \sqrt{\log k^\star}$;

(ii) by setting $k_2 = \left\lfloor \frac{k_1}{c_0\sqrt{\log k_1}} \right\rfloor$, *we also obtain* $\frac{k^\star}{c_1\sqrt{\log k^\star}} \leq k_2 \leq k^\star \leq c_2 k_2 \sqrt{\log k_2}$.

Now we can state our main technical lemma of this section; the proof is based on a technique from [LR88]:

Lemma 7.10. *Let G be a graph of treewidth $k^\star$, U_0 a connected subset of $V(G)$ and $W_0 \subseteq U_0$ with $|W_0| = 4\beta_1 k$, where β_1 is a constant and k a parameter. Then there exists a polynomial-time algorithm that either finds a $\frac{3}{4}$-separator of W_0 in U_0 of size at most $\beta_1 k$; or determines that $k < \frac{4}{3}k^\star\sqrt{\log k^\star}$ and returns a connected subset U of U_0 and a subset $W \subseteq U$ with $|W| \geq 3\beta_1 k$ such that $\alpha^W(U) \geq \frac{1}{\beta_2 k^\star \log k^\star}$, where β_2 is a constant.*

Proof. We denote by $|X|_W$, the number of elements of W in a set X. In our algorithm, we maintain a current component U initialized to U_0, a current set $W \subseteq U$, $W \subseteq W_0$ initialized to W_0, and a current separator S initialized to $\varnothing$. We keep the invariant that $|W| \geq \frac{3}{4}|W_0| = 3\beta_1 k$. In each iteration, we do the following: first, we find a separator (A', B', S') of W in U as guaranteed by Lemma 7.7. Then, we know that $\alpha^W(A', B', S') \leq \beta_0 \alpha^W(U)\sqrt{\log |T|}$, where (A_T, B_T, T) is the optimal $\frac{2}{3}$-separator of W in U. Note that T is at most the size of the optimal $\frac{1}{2}$-separator and hence is at most $k^\star + 1$ by Lemma 7.5. Now, we have

$$\begin{aligned}\frac{|S'|}{|A' \cup S'|_W \cdot |B' \cup S'|_W} &\leq \beta_0 \frac{|T|\sqrt{\log |T|}}{|A_T \cup T|_W \cdot |B_T \cup T|_W} \\ &\leq \beta_1 \frac{k^\star \sqrt{\log k^\star}}{|W|^2},\end{aligned}$$

where the first inequality follows from the fact that T is *some* separator of W in U and thus not sparser than the sparsest separator of W in U; and the second inequality from $|A_T \cup T|_W, |B_T \cup T|_W \geq \frac{1}{3}|W|$ by requiring $\beta_1 \geq 18\beta_0$. It follows that $|S'| \leq \beta_1 k^\star \sqrt{\log k^\star} \frac{|B' \cup S'|_W}{|W|}$. We distinguish two cases:

Case 1: $|S'| > \beta_1 k \frac{|B' \cup S'|_W}{|W_0|}$**.** Then it must be that $k < \frac{4}{3}k^\star\sqrt{\log k^\star}$ and we have

$$\begin{aligned}\alpha^W(A', B', S') &= \frac{|S'|}{|A' \cup S'|_W \cdot |B' \cup S'|_W} \\ &> \frac{\beta_1 k}{|A' \cup S'|_W \cdot |W_0|} \geq \frac{\beta_1 k}{|W_0|^2} \\ &= \frac{\beta_1 k}{16\beta_1^2 k^2} = \frac{1}{16\beta_1 k}\,,\end{aligned}$$

and hence

$$\begin{aligned}\alpha^W(U) &\geq \frac{\alpha^W(A', B', S')}{\beta_0\sqrt{\log |T|}} \\ &\geq \frac{1}{22\beta_0\beta_1 k^\star\sqrt{\log k^\star}\sqrt{\log k^\star + 1}} \geq \frac{1}{\beta_2 k^\star \log k^\star}\,,\end{aligned}$$

for a constant $\beta_2 \geq 44\beta_0\beta_1$.
Case 2: $|S'| \leq \beta_1 k \frac{|B' \cup S'|_W}{|W_0|}$**.** We update our overall separator S to be $S \cup S'$ and check if there exists a connected component U' of $U \setminus S$ that still has more than a $\frac{3}{4}$-fraction of the elements of W_0. If so, we set $U = U'$ and $W = W_0 \cap U$ and repeat our algorithm. Otherwise S is a $\frac{3}{4}$-separator of W_0 in U_0 and we claim that $|S| \leq \beta_1 k$: w.l.o.g we may always assume that $|A' \cup S'|_W \geq |B' \cup S'|_W$ and hence, after each iteration, the set $B' \cup S'$ is discarded. So, the total sum, over all iterations, of the $|B' \cup S'|_W$ is at most $|W_0|$ and the claim follows. □

By setting $s = \beta_1 k$ in Lemma 7.8, we obtain a polynomial-time algorithm that given a graph G and a parameter k, either finds a tree decomposition of G of width at most $5\beta_1 k$ or returns sets U and W as specified in Lemma 7.10. Now, we can apply this algorithm with parameter $k = 2^i$ for $i = 0, 1, 2, \ldots$ to find the first i such that it still fails on i (meaning that a tree decomposition is not constructed) but succeeds in returning a tree decomposition on $i + 1$. Hence, we have

Lemma 7.11. *There is a polynomial-time algorithm that given a graph G of treewidth $k^\star$, returns a number $k \in \mathbb{N}$ with $\frac{k^\star}{10\beta_1} \leq k < \frac{4}{3}k^\star\sqrt{\log k^\star}$ together with a connected subset U of $V(G)$ and a set $W \subseteq U$ with $3\beta_1 k \leq |W| \leq 4\beta_1 k$ such that $\alpha^W(U) \geq \frac{1}{\beta_2 k^\star \log k^\star}$, where β_1, β_2 are constants.*

7.1.2 Randomized Construction of Brambles

Once we are able to find a large set with no sparse cuts in a graph, the rest of the probabilistic proof of Theorem 7.4 (i) in [GM09] becomes algorithmic. Given a set W of vertices, a *concurrent vertex flow of value* ε is a collection of $|W|^2$ flows such that for every ordered pair $(u, w) \in W \times W$, there is a flow of value ε between u and w, and the total amount of flow going through each vertex is at most 1. More precisely, such a flow is defined as follows: for each pair of vertices $(u, w) \in W \times W$, let $\mathcal{P}_{uw}$

be the set of all paths between u and w; a flow between u and w is a function f that assigns a nonnegative flow value $f(P)$ to each path between u and w; we further require that (i) $\sum_{P \in \mathcal{P}_{uw}} f(P) \geq \varepsilon$ for each pair u, w; and (ii) for each vertex $v \in V(G)$, we have $\sum_{v \in V(P), P \in \mathcal{P}_{uw}, (u,w) \in W \times W} f(P) \leq 1$. A maximum concurrent vertex flow can be computed in polynomial time using linear programming techniques [FHL08].

The algorithm FindBramble is given below; steps (2)–(8) are reproduced from the proof of Theorem 7.4 (i) in [GM09]. The basic ideas are as follows: first, we find a number k and sets U and W_0 as in Lemma 7.11; then we compute a maximum concurrent vertex flow on W_0; we select an arbitrary set $W \subseteq W_0$ of size k; afterwards, Grohe and Marx define a certain probability distribution on the paths between the vertices of W, based on the solution to the flow problem, and specify how to randomly pick and combine a number of these paths to construct, with high probability, a bramble $\mathscr{B}$.

Algorithm FindBramble(G)**.**
Input. an arbitrary graph G
Output. a bramble $\mathscr{B}$ in G

1. apply Lemma 7.11 to obtain a number k, and sets $U, W_0 \subseteq V(G)$ as specified;
2. compute a maximum concurrent vertex flow on W_0; let p^{uv} denote the amount of flow that is sent from u to v along a path $P \in \mathcal{P}_{uv}$;
3. select $W \subseteq W_0$ with $|W| = k$ arbitrarily;
4. let $d := \left\lfloor k^{3/2} \right\rfloor$ and $s := \left\lfloor \sqrt{k} \ln k \right\rfloor$; select sets $S_1, \ldots, S_d \subseteq W$, each of size s, uniformly and independently at random; let $S_i = \{u_{i,1}, \ldots, u_{i,s}\}$;
5. for each S_i, select a vertex $z_i \in W - S_i$ at random;
6. for each $(u, v) \in W \times W$, define a probability distribution on $\mathcal{P}_{uv}$ by setting the probability of $P \in \mathcal{P}_{uv}$ to be $\frac{p^{uv}}{\sum_{P' \in \mathcal{P}_{uv}} (p')^{uv}}$;
7. for $i = 1, \ldots, s$ and $j = 1, \ldots, \lfloor \ln n \rfloor$ do
   - select one random path from each of $\mathcal{P}_{z_i,u_{i,1}}, \ldots, \mathcal{P}_{z_i,u_{i,s}}$ according to the probability distribution defined above; let $B_{i,j}$ be the union of these paths;
8. return $\mathscr{B} := \bigcup_{i,j} B_{i,j}$.

Note that all the steps of the algorithm can be performed in polynomial time; in particular, the p^{uv} are also variables in the linear programming formulation of the maximum concurrent flow problem and only a polynomial number of them will have nonzero value (cf. [FHL08]).

Lemma 7.12 (adapted from Grohe and Marx [GM09])**.** *With probability at least* $1 - 1/k$*, the set* $\mathscr{B}$ *constructed above is a bramble. With probability at least* $1 - 1/n$*, the order of this bramble is at least* $\frac{k^{3/2} \alpha^{W_0}(U)}{\beta_3 \ln k \ln |W_0|}$*, for a constant* β_3*.*

Theorem 7.13. *There exists a randomized polynomial time algorithm that, given a graph* G *of treewidth* $k^\star$*, constructs with high probability a bramble in* G *of order* $\Omega(\frac{\sqrt{k^\star}}{\ln^3 k^\star})$ *and size* $\mathcal{O}(k^{\star 3/2} \ln k^\star \ln n)$*.*

Proof. We apply the algorithm FindBramble and use Lemma 7.11 and Lemma 7.12 to bound the order and size of the bramble. For the size of the bramble, we know that $|\mathscr{B}| = \left\lfloor k^{3/2} \right\rfloor \lfloor \ln n \rfloor = O((k^\star \sqrt{\log k^\star})^{3/2} \ln n) = O(k^{\star^{3/2}} \ln k^\star \ln n)$. The order is at least

$$\geq \frac{k^{3/2} \cdot \alpha^{W_0}(U)}{\beta_3 \ln k \ln |W_0|} \geq \frac{k^{\star^{3/2}}}{\beta_4 k^\star \ln k^\star \ln^2 k} \geq \frac{\sqrt{k^\star}}{\beta_5 \ln^3 k^\star},$$

for appropriate constants $\beta_4, \beta_5 > 0$. □

Note that by a slight modification of the algorithm above, one can also construct a bramble of size $\mathcal{O}(k^{\star^{3/2}} \ln n)$ and order $\Omega(\frac{\sqrt{k^\star}}{\ln^4 k^\star})$.

7.1.3 Weak k-Webs

Definition 7.14 (Weak k-Web)**.** A *weak k-web of order h* in a graph G is a set of h disjoint trees $T_1, \ldots, T_h$ such that for all $1 \leq i < j \leq h$ there is a set $\mathcal{P}_{i,j}$ of k disjoint paths connecting T_i and T_j. If the trees $T_1, \ldots, T_h$ are all paths, we denote the resulting structure by a *weak k-web of paths of order h*.

In [RW08], it is shown that any bramble of order at least hk, contains a weak k-web of paths of order h. They use this structure to show the existence of grid-like minors. Even though we provide a different proof for grid-like minors, we still include the following lemma as it might be of independent interest:

Lemma 7.15. *There is a polynomial-time algorithm that given a bramble $\mathscr{B}$ of order at least $chk\sqrt{\log k}$ in a graph G, computes a weak k-web of paths of order h in G, where c is a constant.*

Proof. First, as in [BBR07, RW08], we observe that one can find a simple path P in G that hits every element of $\mathscr{B}$ by a simple greedy algorithm: suppose by induction, that we have already constructed a path P' that hits some elements of $\mathscr{B}$ and that there is one element $B \in \mathscr{B}$ that intersects P' in only an endpoint v. If there is an element $B' \in \mathscr{B}$ that is not hit by P', we extend P' by a path $P_{vu} \subseteq B$, such that $P_{vu} \cap B' = \{u\}$; this is always possible, since B and B' touch. Furthermore P_{vu} is otherwise disjoint from P' and the extended path intersects B' in only one vertex. Hence, our claim follows by induction.

Now, we move on P from left to right and at each vertex v, we consider the subpath P_v and the subbramble $\mathscr{B}_v \subseteq \mathscr{B}$ that is hit by P_v. We can use the duality of brambles and tree decompositions and Corollary 7.9 to find a number k_v, such that $k_v \leq k_v^\star \leq c' k_v \sqrt{\log k_v}$, where $k_v^\star$ is the order of $\mathscr{B}_v$ and c' is a constant. Now, let uv be an edge of P such that $k_u < k \leq k_v$. Note that $k_v^\star \leq k_u^\star + 1$, and hence we obtain that the order of the subbramble $\mathscr{B}_v$ is at least k and at most $ck\sqrt{\log k}$, for a properly defined constant c. We set $P_1 = P_v$ and $P' = P - P_1$ and $\mathscr{B}' = \mathscr{B} \setminus \mathscr{B}_v$ and iterate this process on P' and $\mathscr{B}'$. Since the order of the bramble $\mathscr{B}_v$ that is cut away in each iteration is at most $ck\sqrt{\log k}$ and since the order of $\mathscr{B}$ is at least $chk\sqrt{\log k}$, we indeed obtain at least h

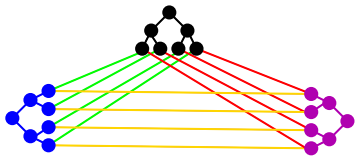


Figure 7.2: Schematic illustration of a 4-web of order 3.

disjoint paths $P_1, \dots, P_h$ and brambles $\mathscr{B}_1, \dots, \mathscr{B}_h$, each of order at least k, such that for all i, P_i hits $\mathscr{B}_i$ and for $i < j$, P_i does not intersect any element of $\mathscr{B}_j$. Reed and Wood [RW08] show that in this case, there exist at least k disjoint paths between P_i and P_j for each $i < j$, and hence the lemma is proven. □

Corollary 7.16. *For any $\varepsilon > 0$, there is a constant c such that if for a graph G, we have $\operatorname{tw}(G) \geq ch^{2+\varepsilon}k^{2+\varepsilon}$, then G contains a weak k-web of paths of order h that can be constructed in randomized polynomial time.*

7.1.4 k-Webs

In the definition of a weak k-web, we allow that the disjoint paths that connect two of the trees T_i and T_j intersect some other tree(s) T_ℓ. It turns out that if we can manage to keep the trees disjoint from the paths interconnecting them, we obtain a structure that is sometimes much more useful. This leads us to the definition of *k-webs* below; but first we need some preparation:

Definition 7.17. A tree T is *subcubic* if its maximum degree is at most 3. A set $X \subseteq V(T)$ is called *flat* if every vertex $v \in X$ has degree at most 2 in T.

A slight adaption of Fredericson's FindClusters algorithm [Fre85] gives us the following simple lemma:

Lemma 7.18. *Let T be a subcubic tree and $X \subseteq V(T)$ be a set of $2k\ell$ vertices, where $k, \ell \in \mathbb{N}$. Then there are ℓ disjoint subtrees $T_1, \dots, T_\ell$ of T such that $|X \cap V(T_i)| = k$, for all $1 \leq i \leq \ell$.*

Definition 7.19 (k-Web)**.** A *k-web of order h* in a graph G is a collection of subgraphs $(T, (T_i)_{1\leq i\leq h}, (A_i)_{1\leq i\leq h}, B)$ of G such that

(i) T is a subcubic tree and $V(B \cap T) = \bigcup_{1\leq i\leq h} V(A_i)$;

(ii) $T_1, \dots, T_h$ are disjoint subtrees of T and for $1 \leq i \leq h$, $A_i \subseteq T_i$ is flat in T;

(iii) for $1 \leq i < j \leq h$ there is a set $\mathcal{P}_{i,j}$ of k disjoint paths in B connecting A_i and A_j.

See Figure 7.2 for an illustration of a k-web. Note that the main restriction of a k-web compared to a weak k-web is that the paths $\mathcal{P}_{i,j}$ are required to be disjoint from the trees $T_1, \dots, T_h$ (except for their endpoints); on the other hand, the advantage of a weak

k-web of paths is that all its trees are paths. The notion of a k-web is similar to the notion of a *k-mesh* introduced by Diestel et al. [DGJT99] with the difference that in a k-mesh, we have a tree and a collection of vertices of the tree such that there exist disjoint paths between *each two k-element subsets of these vertices*; but in a k-web, we have predefined subsets of k-vertices and only require to have paths between these sets. It is this distinction that makes k-webs efficiently computable. Adapting a proof by Diestel et al. [DGJT99, Die05] we show that any graph of large enough treewidth contains a k-web of large order that can be computed in polynomial time.

Lemma 7.20 (adapted from Diestel et al. [DGJT99])**.** *Let $h, k \geq 1$ be integers. If G has treewidth at least $(2 \cdot h+1) \cdot k-1$ then G contains a k-web of order h. Furthermore, there is a polynomial time algorithm which, given G, k, h either computes a tree decomposition of G of width at most $(2 \cdot h + 1) \cdot k - 2$ or a k-web of order h in G.*

Proof. W.l.o.g. we assume that G is connected. Let $l := 2 \cdot k \cdot h$. A *pre-web* is a collection $\mathcal{W} := (U, \mathcal{D}, \{T_C : C \text{ is a component of } G - U\})$ where $U \subseteq V(G)$, $\mathcal{D} := (D, (B_t)_{t \in V(D)})$ is a tree decomposition of $G[U]$ of width at most $l + k - 2$ and for each component C of $G - U$, T_C is a subcubic tree in $G - C$ such that

(i) there is a bag B of $\mathcal{D}$ with $N(C) \subseteq B$;
(ii) $N(C)$ is a flat subset of $V(T_C)$;
(iii) T_C has a leaf in $N(C)$ or $|T_C| = 1$ and $T_C \subseteq N(C)$.

U is called the domain of the pre-web. The order of $\mathcal{W}$ is $|U|$. Inductively, we will construct a sequence of pre-webs of growing order until we either find a k-web of order h or a pre-web with domain $V(G)$ and hence a tree decomposition of G of width at most $l + k - 2$.

To initialize the algorithm choose a vertex $v \in V(G)$ and let $U := \{v\}$, $\mathcal{D} := \{\{0\}, B_0 := \{v\}\}$ and $T_C := v$ for each component C of $G - v$. Clearly, $(U, \mathcal{D}, \{T_C : C$ component of $G - v\})$ is a pre-web.

Suppose we have already constructed a pre-web $(U, \mathcal{D}, \{T_C : C \text{ component of } G - U\})$. If $U = V(G)$ we are done. Otherwise, let C be a component of $G - U$ and let $T := T_C$. By assumption, there is a node $t \in V(D)$ with bag B_t, where D is the tree underlying $\mathcal{D}$, such that $X := N(C) \subseteq B_t$.

If $|X| \leq l$ then let v be a leaf of T in X, which exists by assumption. Let $u \in V(C)$ be a neighbor of v and set $U' := U \cup \{u\}$. Let $T' := T + \{u, v\}$ be the tree obtained from T by adding u as a new vertex joined to v. Further, let $\mathcal{D}'$ be the tree decomposition of $G[U']$ obtained from $\mathcal{D}$ by adding a new vertex s with bag $B_s := X \cup \{u\}$ joined to t in $\mathcal{D}'$. Now let C' be a component of $G - U'$. If $C' \cap C = \varnothing$ set $T'_{C'} := T_{C'}$. Otherwise, $C' \subseteq C$ and we set $T'_{C'}$ to be the minimal subtree of T' containing $N(C')$. By construction, $N(C')$ contains v. Further, as $X = N(C)$ was flat in T, $N(C')$ is flat in $T'_{C'}$. Hence, $(U', \mathcal{D}', \{T'_{C'} : C'$ component of $G - U'\})$ is a pre-web of order $|U| + 1$.

Now suppose $|X| = l$. Let $T_1, \ldots, T_h$ be a collection of disjoint subtrees of T with $|V(T_i) \cap X| = k$, which exist by Lemma 7.18, and let $A_i := V(T_i) \cap X$. For each $1 \leq i < j \leq h$ compute a maximal set $\mathcal{P}_{i,j}$ of disjoint paths in $H := G[V(C) \cup$

$A_i \cup A_j] - E(G[A_i \cup A_j])$ joining A_i and A_j. If all $P_{i,j}$ contain at least k paths then $(T, (T_i)_{1\leq i\leq h}, (A_i)_{1\leq i\leq h}, C \cup N(C))$ is a k-web of order h and we are done. Otherwise, let A_i, A_j be such that $k' := |\mathcal{P}_{i,j}| < k$. By Menger's theorem, there is a set $S \subseteq V(H)$ of k' vertices separating A_i, A_j in H. Clearly, S contains one vertex of each $P \in \mathcal{P}_{i,j}$. We denote by $P_s \in \mathcal{P}_{i,j}$ the path containing $s \in S$.

Let $X' := X \cup S$ and $U' := U \cup S$ and let $\mathcal{D}'$ be the tree decomposition of $G[U']$ obtained from $\mathcal{D}$ by adding a new vertex r with bag X' joined to t. By construction, $|X'| \leq |X|+|S| \leq l+k-1$. Let C' be a component of $G-U$. If $C' \cap C = \varnothing$ set $T'_{C'} := T_{C'}$. Otherwise, $C' \subseteq C$ and $N(C') \subseteq X'$. Furthermore, C' must have at least one neighbor v in $S \cap C$ since X does not separate C' from $S \cap C$. By construction of S, C' cannot have neighbors in both $A_i \setminus S$ and $A_j \setminus S$. W.l.o.g. we assume that $N(C') \cap A_i = \varnothing$. Let $T'_{C'}$ be the union of T_C with all $A_i - S$-subpaths of P_s for $s \in C \cap N(C')$. As these subpaths start in $A_i \setminus S$ and have no inner vertices in X', they do not meet C'. We claim that $\mathcal{W}' := (U', \mathcal{D}', \{T'_{C'} : C' \text{ component of } G - U'\})$ is a pre-web. Clearly, $\mathcal{D}'$ is a tree decomposition of $G[U']$ of width at most $l + k - 2$. Furthermore, each tree $T'_{C'}$ is clearly subcubic. Now let C' be a component of $G - U'$. If $C' \cap C = \varnothing$, then C' is also a component of $G - U$ and hence $T'_{C'} = T_{C'}$ and therefore there is a bag B_t in $\mathcal{D}$ with $N(C') \subseteq B_t$ and the additional conditions on $T_{C'}$ are met. Otherwise, $N(C') \subseteq X'$. Let $T := T'_{C'}$. Then T contains a leaf in X' (the vertex v constructed above). The degree conditions imposed on T are clearly met as well. Furthermore, $N(C')$ is a terminal subset of $T'_{C'}$. It follows that $\mathcal{W}'$ is a pre-web of order $|U'| > |U|$.

Obviously, the algorithm takes only a linear number of steps. Furthermore, each step can be computed in polynomial time. This concludes the proof. □

Lemma 7.21. *Let $k \geq 1$. If G contains a $(k+1)$-web of order $k+1$ then the treewidth of G is at least k.*

Proof. Let $(T, (T_i)_{1\leq i\leq k+1}, (A_i)_{1\leq i\leq k+1}, Z)$ be a $(k+1)$-web of order $k+1$ in G. Towards a contradiction, assume G has a tree decomposition $(D, (B_t)_{t\in V(D)})$ of width $< k$. For an edge $st \in E(D)$, we denote by D_{s-t} the subtree of $D - st$ that contains s and by $B(D_{s-t})$, the union of the bags of D_{s-t}. We orient the edges of D as follows. If $st \in E(D)$, let $I_s := \{T_i : T_i \subseteq B(D_{s-t})\}$ and define I_t analogously; we orient the edge towards s if $|I_s| \geq |I_t|$ and otherwise orient the edge towards t. As D is acyclic, there must be a node $s^\star \in V(D)$ such that all incident edges are oriented towards $s^\star$. Now, for each edge $s^\star t \in E(D)$, $B(D_{s^\star-t})$ contains at least one T_i completely; on the other hand, as $|B_{s^\star}| \leq k$, $B_{s^\star}$ can not contain a vertex of every T_i and there must be an edge $s^\star t^\star \in E(D)$ such that $B(D_{t^\star-s^\star})$ also contains some T_i completely. Let $T_i \subseteq B(D_{s^\star-t^\star})$ and $T_j \subseteq B(D_{t^\star-s^\star})$; but then, there are $k+1$ disjoint paths between T_i and T_j and each of these must have an inner vertex in $B_{s^\star} \cap B_{t^\star}$, which is impossible. □

Corollary 7.22. *There is a polynomial time algorithm which, given a graph G either computes a $(k+1)$-web of order $k+1$ and thereby proves that $\mathrm{tw}(G) \geq k$ or a tree decomposition of G of width $\mathcal{O}(k^2)$.*

7.1.5 Constructing a Bramble from a k-Web

In this subsection, we briefly mention an alternative bramble construction that differs from Section 7.1.2 in that its *size* does not involve n but instead, its *order* is less[2].

Lemma 7.23. *Given a k^2-web of order k, one can construct a bramble of size k^3 and order k.*

Proof. Let $(T, (T_i)_{1\leq i\leq k}, (A_i)_{1\leq i\leq k}, B)$ be a k^2-web of order k and $\mathcal{P}_{i,j} = \{P_{ij}^1, \ldots, P_{ij}^{k^2}\}$ be the k^2 disjoint paths between A_i and A_j. Let $\widehat{P_{ij}^t}$ be the path P_{ij}^t without the last edge that connects it to A_j. Define $B_i^t = T_i \cup \bigcup_{j=1}^{k} \widehat{P_{ij}^t}$, for $1 \leq i \leq k$ and $1 \leq t \leq k^2$, and let $\mathscr{B} = \bigcup_{i,t} B_i^t$. Then $\mathscr{B}$ is clearly a bramble of size k^3. Suppose there is a hitting set of $\mathscr{B}$ of order less than k; then there is an i, such that T_i is not covered. Hence, for $1 \leq t \leq k^2$, B_i^t must be covered using vertices in $\bigcup_{t,j} \widehat{P_{ij}^t}$; but note that any vertex in this union has degree at most k, and so at least k vertices are needed to cover all these k^2 sets. □

Theorem 7.24. *There exists a polynomial time algorithm that, given a graph G of treewidth $k^\star$, constructs a bramble in G of size $O(k^\star)$ and order $\Omega((\frac{k^\star}{\sqrt{\log k^\star}})^{1/3})$.*

Proof. By Corollary 7.9, we can compute an integer k_2 with $\frac{k^\star}{c_1\sqrt{\log k^\star}} \leq k_2 \leq k^\star$. We set $k = \frac{k_2^{1/3}}{2}$ and use Lemma 7.20 to obtain a k^2-web of order k in G. Our claim then follows by Lemma 7.23. □

7.2 Constructing Grid-Like Minors

Let $\mathcal{P}$ and $\mathcal{Q}$ each be a set of disjoint connected subgraphs of a graph G. We denote by $\mathcal{I}(\mathcal{P}, \mathcal{Q})$ the *intersection graph* of $\mathcal{P}$ and $\mathcal{Q}$ defined as follows: $\mathcal{I}(\mathcal{P}, \mathcal{Q})$ is the bipartite graph that has one vertex for each element of $\mathcal{P}$ and $\mathcal{Q}$ and an edge between two vertices if the corresponding subgraphs intersect.

Definition 7.25 (Grid-Like Minor)**.** Let $\mathcal{P}$ and $\mathcal{Q}$ be each a set of disjoint paths in a graph G. $(\mathcal{P}, \mathcal{Q})$ is called a *grid-like minor of order ℓ* in G if $\mathcal{I}(\mathcal{P}, \mathcal{Q})$ contains the complete graph $\mathbf{K}_\ell$ as a minor. If the $\mathbf{K}_\ell$-minor is, in fact, a topological minor, we call the structure a *topological grid-like minor of order ℓ*.

Theorem 7.26 (Reed and Wood [RW08])**.** *There exists a constant c such that every graph with treewidth at least $c\ell^4\sqrt{\log \ell}$ contains a grid-like minor of order ℓ. Conversely, every graph that contains a grid-like minor of order ℓ has treewidth at least $\left\lceil \frac{\ell}{2} \right\rceil - 1$.*

The proof given in [RW08] is existential and proceeds as follows: first, using a large bramble, a weak k-web of paths is constructed; then for each pair of sets of disjoint paths

[2]The existence of such a bramble is briefly mentioned in [GM09] but it is not presented; I would like to thank Dániel Marx for a helpful discussion on this matter.

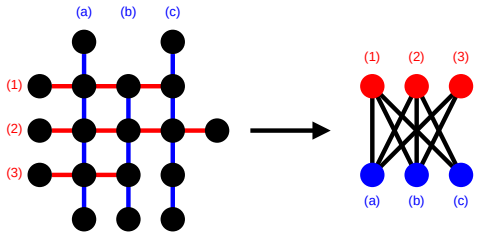


Figure 7.3: A grid-like minor of order 3; (a) two sets of disjoint paths: blue vertical ones and red horizontal ones; (b) their intersection graph is bipartite and includes $\mathbf{K}_3$ as a minor.

in the k-web, it is checked whether their union contains a grid-like minor of large order; if this is not true for any pair, one can obtain a grid-like minor using the Lovász Local Lemma. In this section, we make their proof algorithmic by showing how the individual major steps of the proof can be performed in polynomial time. We show

Theorem 7.27. *There are constants c_1, c_2, c_1', and c_2' such that if a graph G has*

(i) $\operatorname{tw}(G) \geq c_1\ell^5$, *then G contains either a model of $\mathbf{K}_\ell$ or a topological grid-like minor $(\mathcal{P}, \mathcal{Q})$ of order ℓ together with a k-web of order t, $(T, (T_i)_{1\leq i\leq t}, (A_i)_{1\leq i\leq t}, B)$, where $k = \mathcal{O}(\ell^6)$, t is either 3 or 4, $\mathcal{P}$ is a set of disjoint paths between T_1 and T_2, and $\mathcal{Q}$ is a set of disjoint paths between T_i and T_j with $(i, j) = (1, 3)$ for $t = 3$ and $(i, j) = (3, 4)$ for $t = 4$.*

(ii) $\operatorname{tw}(G) \geq c_2\ell^8$, *$G$ contains a topological grid-like minor of order ℓ.*

Furthermore, the corresponding objects can be constructed by a randomized algorithm with expected polynomial running time. If the bounds on the treewidth are loosened to $c_1'\ell^7$ and $c_2'\ell^{12}$, respectively, then a deterministic algorithm can be used.

The first step of the proof in [RW08] is to find a weak k-web of paths; instead, we make use of a k-web as described in Section 7.1.4. We proceed with the second main step of the algorithm.

7.2.1 Finding Complete Topological Minors

Once we have a k-web, we need to determine if the intersection graph of any pair of the disjoint paths contains a large complete graph as a minor. Thomason [Tho01] showed that if the average degree of a graph is at least $cp\sqrt{\log p}$, then the graph contains $\mathbf{K}_p$ as a minor (and that this bound is tight). His proof is very complicated and it is not clear if it can be turned into a polynomial-time algorithm. However, if we are looking for a *topological minor*, the necessary and sufficient bound is cp^2 as shown by Bollobás and Thomason [BT98]. Furthermore, it turns out that their proof is algorithmic:

Theorem 7.28 (adapted from Bollobás and Thomason [BT98])**.** *If a graph G has average degree at least cp^2, for a constant c, then G contains $\mathbf{K}_p$ as a topological minor. Furthermore, a model of $\mathbf{K}_p$ can be found in G in polynomial time.*

Note that by the definition of a grid-like minor, we do not necessarily need a topological minor but we use them for two reasons: first, we know we can compute them in polynomial time; second, it is much more convenient to work with topological minors in MSO_2 in Chapter 8. The algorithm for Theorem 7.28 is given by Algorithm TopMinor below. We refer for the full proof of correctness to the original paper [BT98] and just argue briefly that each of the steps can be performed in polynomial time. We call a set of vertices *linkable* if for any pairing of its elements, there exist disjoint paths between the given pairs. A graph is said to be (k, ℓ)-*linked* if every set of k vertices contains a subset of size ℓ which is linkable.

The first step of the algorithm is due to a theorem of Mader [Mad72] (see also [Die05, Theorem 1.4.3]) and can be computed as follows: we select G_1 as a minimal subgraph G, such that $|V(G_1)| \geq 256p^2$ and $|E(G_1)| \geq 256p^2(|V(G_1)|-128p^2)$; we can start by setting $G_1 = G$ and deleting vertices and edges and finding minimum cuts to reduce G_1 as long as the desired properties are still satisfied. Clearly, these operations can all be performed in polynomial time and Mader shows that in the end, G_1 will be $128p^2$-connected.

Algorithm TopMinor(G, p)**.**
Input. a graph G with $|E(G)| \geq 256p^2n$
Output. a topological minor $\mathbf{K}_p$ in G
(in the following, the index i ranges appropriately)

1. find a subgraph G_1 of G that is at least $128p^2$-connected;
2. select an arbitrary set $X = \{x_1, \ldots, x_{3p}\}$ in G_1 and let $G_2 = G_1 - X$;
3. let H_1 be a minimal minor of G_2 subject to $|E(H_1)| \geq 63p^2|V(H_1)|$;
4. let z be a vertex of H_1 of minimum degree and $H_2 := H_1[N(z)]$;
5. let H_3 be a minimal minor of H_2 subject to $|V(H_3)| \geq 23p^2$ and $|E(H_3)| \geq \frac{23p^2}{2}(\log \frac{|V(H_3)|}{23p^2} + 1)|V(H_3)|$;
6. let u be a vertex of minimal degree of H_3 and $H := H_3[N(u)]$;
7. select $3p$ arbitrary disjoint sets $Y_1, \ldots, Y_{3p}$ in G_2 each of size $5p$ such that Y_i consists of neighbors of x_i;
8. using the model of H in G, find a set $Z \subseteq \bigcup Y_i$ of size $7p^2$ which is linkable;
9. let $Z_i = Z \cap Y_i$ and select indices $j_1, \ldots, j_p$ such that $|Z_{j_i}| \geq p - 1$;
10. return $\{x_{j_1}, \ldots, x_{j_p}\}$ and the disjoint paths that exist between the Z_{j_i}.

The graphs H_1 and H_3 can easily be constructed by starting with G_2 (respectively with H_2) and contracting and deleting edges as long as the given conditions are satisfied. The only major difficult step of the algorithm, is the 8th step. By using the model of H in G, Bollobás and Thomason show that G_2 is $(15p^2, 7p^2)$-linked, and hence that the set Z exists. In order to do so, they use Menger's theorem to find disjoint paths from $\bigcup Y_i$ to the model of H in G and show that at least half the paths can be modified in such a way that they are linkable through the model of H. The modifications are of a simple nature and we refer to [BT98, Theorem 1] for the details. Since the application of Menger's theorem amounts to a maximum-flow computation, all of the steps can indeed be performed in polynomial time.

7.2.2 Algorithmic Application of the Lovász Local Lemma

Recall that a graph G is called *d-degenerate* if every subgraph of G has a vertex of degree at most d and note that Theorem 7.28 implies that if G does not contain $\mathbf{K}_p$ as a topological minor, then G is cp^2-degenerate, for a constant c. Reed and Wood proved the following lemma, where e denotes the base of the natural logarithm:

Lemma 7.29 (Reed and Wood [RW08])**.** *For some $r \geq 2$, let $V_1, \ldots, V_r$ be the color classes in an r-coloring of a graph H. Suppose that $|V_i| \geq n := 2e(2r-3)d$ for all $1 \leq i \leq r$ and assume $H[V_i \cup V_j]$ is d-degenerate for $1 \leq i < j \leq r$. Then there exists an independent set $\{x_1, \ldots, x_r\}$ of H, such that each $x_i \in V_i$.*

The proof of this lemma in [RW08] is *existential* and uses the Lovász Local Lemma (LLL) [EL75]. Reed and Wood note that if $n \geq r(r-1)d+1$, a simple minimum-degree greedy algorithm will find such an independent set, and pose as an open question if this algorithmic bound can be improved. Very recently, Moser [Mos09] and Moser and Tardos [MT09] proved in their breakthrough work that the LLL can be actually made algorithmic by a randomized algorithm with expected polynomial running time. Hence, we obtain that there exists such a randomized algorithm with expected polynomial running time that finds the independent set specified by Lemma 7.29.

Corollary 7.30. *For some $r \geq 2$, let $V_1, \ldots, V_r$ be the color classes in an r-coloring of a graph H. Suppose that $|V_i| \geq n := 2e(2r-3)d$ for all $1 \leq i \leq r$ and assume $H[V_i \cup V_j]$ is d-degenerate for $1 \leq i < j \leq r$. Then there exists a randomized algorithm that finds an independent set $\{x_1, \ldots, x_r\}$ of H with $x_i \in V_i$ for all $1 \leq i \leq r$, in expected time polynomial in n. Furthermore, if, instead, we have $n \geq r(r-1)d+1$, then a deterministic polynomial-time algorithm can be used.*

7.2.3 Putting Things Together

Starting with a (weak) k-web of order h, we consider the disjoint paths $\mathcal{P}_{i,j}$ between the pairs of trees from the web; note that these paths can be found by a simple max-flow computation in polynomial time. For each pair of these paths, we check if the average degree of the intersection graph is large; if so, we find a topological grid-like minor by Theorem 7.28; otherwise, we consider the intersection graph $\mathcal{I}$ of *all* the $r := \binom{h}{2}$ sets of paths; i.e. $\mathcal{I}$ is an r-partite graph, having a vertex for each path out of $\mathcal{P}_{i,j}$, for $1 \leq i < j \leq h$, and an edge between two vertices if the corresponding paths intersect. Now we can invoke Corollary 7.30 with $\mathcal{I}, r$ and $d := c_1 p^2$. We obtain

Lemma 7.31. *Let G be a graph and let $T_1, \ldots, T_h$ be the disjoint trees of a (weak) k-web of order h in G with $k \geq ch^2p^2$, for a suitable constant c. Then there exists a randomized algorithm with polynomial expected running time that finds in G either a topological grid-like minor of order p or a set of $\binom{h}{2}$ disjoint paths $Q_{ij}, 1 \leq i < j \leq h$, where Q_{ij} is a path connecting T_i and T_j. If $k \geq c'h^4p^2$, for a suitable constant c', a deterministic polynomial-time algorithm also exists.*

By using the k-web of order h that is guaranteed by Lemma 7.20 and setting $k = ch^2p^2$, we immediately obtain a randomized algorithm that given a graph G of treewidth at least ch^3p^2 computes in G either a model of $\mathbf{K}_h$ or a topological grid-like minor of order p; a deterministic variant is obtained if $\mathrm{tw}(G) \geq c'h^5p^2$. This observation, in turn, immediately proves Theorem 7.27 (i) by setting $h = p = \ell$. As for claim (ii), we set $h = \ell^2$ and $p = \ell$, and hence obtain either a topological grid-like minor of order ℓ or a model of $\mathbf{K}_{\ell^2}$; but then, consider the following: let H be a graph that consists of ℓ "horizontal" paths and $\binom{\ell}{2}$ "vertical" edges, one connecting each pair of the horizontal paths. Then H has less than ℓ^2 vertices, has maximum degree 3, and any subdivision of H is a topological grid-like minor of order ℓ; now, any graph that has $\mathbf{K}_{\ell^2}$ as a minor, has H as a topological minor, and hence contains a topological grid-like minor of order ℓ (recall that if a graph H has maximum degree 3 and is a minor of a graph G, then it is also a topological minor of G).

Note that by using the weak k-web of paths that is given by Corollary 7.16, one can also directly obtain a topological grid-like minor of order h but the bounds would be worse than those obtained in Theorem 7.27.

7.3 Perfect Brambles and a Meta-Theorem

In this section, we define perfect brambles and show that certain parameterized problems can be decided efficiently using this notion as a replacement for grid-minors.

Definition 7.32 (Perfect Bramble)**.** A bramble $\mathscr{B}$ in a graph G is called *perfect* if

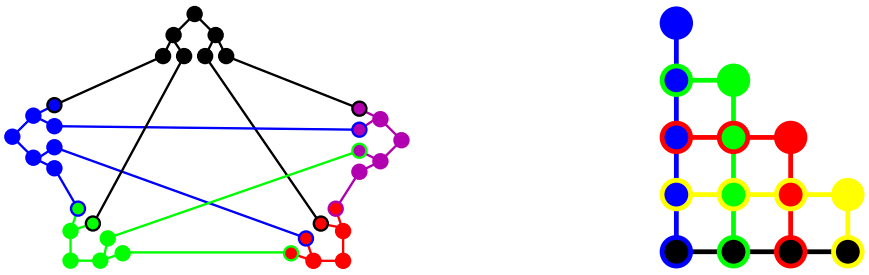
1. any two $B, B' \in \mathscr{B}$ intersect;
2. for every $v \in V(G)$ there are at most two elements of $\mathscr{B}$ that contain v;
3. every vertex has degree at most 4 in $\bigcup \mathscr{B}$.

See Figure 7.4 for some examples of perfect brambles. Perfect brambles have some interesting properties, such as the ones given below.

Lemma 7.33. *Let $\mathscr{B} = \{B_1, \ldots, B_\ell\}$ be a perfect bramble and $H = \bigcup \mathscr{B}$. Then we have*

(i) every element $B \in \mathscr{B}$ has at least $\ell - 1$ vertices;
(ii) every element $B \in \mathscr{B}$ has at least $\ell - 2$ edges that do not appear in any other element of $\mathscr{B}$;
(iii) H has at least $\frac{\ell(\ell-1)}{2}$ vertices and at least $\ell(\ell-2)$ edges;
(iv) the order of $\mathscr{B}$ is exactly $\left\lceil \frac{\ell}{2} \right\rceil$ *and hence, can be computed in linear time;*
(v) the treewidth of H is at least $\left\lceil \frac{\ell}{2} \right\rceil - 1$.

Proof. Claim (i), (ii), and (iii) follow from the fact that B intersects $\ell - 1$ other elements of $\mathscr{B}$ and because of property (2) in Definition 7.32, an extra vertex is needed for each; also, at least $\ell - 2$ edges are needed to connect these at least $\ell - 1$ parts of $\mathscr{B}$ together. Since each vertex covers at most two elements of $\mathscr{B}$, at least $\frac{\ell}{2}$ vertices are needed for a complete hitting set; on the other hand, since each two elements of $\mathscr{B}$ meet at a vertex, $\frac{\ell}{2}$ vertices are also sufficient. This proves (iv), and by Theorem 7.2 also (v). □



(a) (b)

Figure 7.4: Two examples of perfect brambles. The bramble elements are indicated with different colors; vertices with different boundary and inner color belong to two bramble elements.

Using Theorems 7.26 and 7.27, we obtain

Theorem 7.34. *There are constants c_1, c_2, c_3, such that for any graph G, we have*

(i) if $\mathrm{tw}(G) \geq c_1 \ell^4 \sqrt{\log \ell}$, then G contains a perfect bramble of order ℓ;

(ii) if $\mathrm{tw}(G) \geq c_2 \ell^5$, there is randomized algorithm with expected polynomial running time that finds a perfect bramble of order ℓ in G;

(iii) if $\mathrm{tw}(G) \geq c_3 \ell^7$, a deterministic algorithm for the same purpose exists.

Proof. Consider a grid-like minor of order 2ℓ in G; let $\mathcal{P}, \mathcal{Q}$ be the sets of disjoint paths such that $\mathcal{I} = \mathcal{I}(\mathcal{P}, \mathcal{Q})$ contains $\mathbf{K}_{2\ell}$ as a minor. Let $I_1, \ldots, I_{2\ell}$ be the connected subgraphs of $\mathcal{I}$ that define a model of $\mathbf{K}_{2\ell}$. For each of these subgraphs I_j, we define a subgraph B_j of G that consists of the set of paths out of $\mathcal{P}$ and $\mathcal{Q}$ that are contained in I_j. Then $\mathscr{B} = \{B_1, \ldots, B_{2\ell}\}$ is a perfect bramble of order ℓ; this can be checked straightforwardly by noting that (i) $\mathcal{P}$ and $\mathcal{Q}$ are each a set of disjoint paths; (ii) the sets $I_1, \ldots, I_{2\ell}$ are disjoint in $\mathcal{I}$ and there is an edge between any two of them; (iii) when there is an edge between two sets I_i and I_j, it means that there is a path in B_i and a path in B_j, one from $\mathcal{P}$ and one from $\mathcal{Q}$, such that these two intersect[3].

On the other hand, consider a model of $\mathbf{K}_{2\ell}$ in G that is obtained as one possible outcome of applying Theorem 7.27 (i), which, in turn, is obtained from Lemma 7.31. It consists of a number of subcubic trees $T_1, \ldots, T_{2\ell}$ and a number of disjoint paths $Q_{ij}, 1 \leq i < j \leq 2\ell$. For $1 \leq i \leq 2\ell$, we define a set B_i to be the union of T_i with "the first half" of each of the paths $Q_{ij}, 1 \leq j \leq \ell, j \neq i$, where "the first half" is defined as follows: for each path Q_{ij}, we select an arbitrary vertex v_{ij} on Q_{ij}; the first half of a path Q_{ij}, starting at the tree T_i, is then the part of the path up to and including v_{ij}. Then, one can easily check that $\mathscr{B} = \{B_1, \ldots, B_{2\ell}\}$ is a perfect bramble of order ℓ.

Now our claim follows by invoking Theorems 7.26 and 7.27, respectively. □

[3] A similar proof is also given in [RW08].

Corollary 7.35. *For any graph G of treewidth k, there exists a subgraph H of G with treewidth polynomial in k and maximum degree 4. Furthermore, H can be computed in polynomial time.*

An interesting consequence of this corollary is that if the relation between treewidth and grid-minors is indeed polynomial (see Theorem 7.3), then it suffices to prove it only for graphs of bounded degree, in fact, only for perfect brambles.

Next, let $\mathscr{G}$ denote the set of all graphs; we obtain the following *meta theorem*:

Theorem 7.36. *Let $c, \alpha > 0$ be constants, G be a graph, and $\pi : \mathscr{G} \to \mathbb{N}$ be a parameter, such that*

(i) if H is a subgraph of G, then $\pi(H) \leq \pi(G)$;

(ii) on any graph $H = \bigcup \mathscr{B}$, where $\mathscr{B}$ is a perfect bramble of order ℓ, $\pi(H) \geq c\ell^{\alpha}$;

(iii) given a tree decomposition of width ℓ on a graph H, $\pi(H)$ can be computed in time $\mathcal{O}(2^{\mathrm{poly}(\ell)}\mathrm{poly}(n))$;

then there is an algorithm with running time $\mathcal{O}(2^{\mathrm{poly}(k)}\mathrm{poly}(n))$ that decides if $\pi(G) \leq k$. Furthermore, if in (i), (ii), and (iii) above, a corresponding witness can be constructed in time $\mathcal{O}(2^{\mathrm{poly}(k)}\mathrm{poly}(n))$, then a witness, proving or disproving $\pi(G) \leq k$, can also be constructed in the given time.

Proof. If the treewidth of G is large enough, i.e. at least $c'k^{4/\alpha}\sqrt{\log k}$, for a suitable constant c', then, by Theorem 7.34 (i), G contains a perfect bramble of order $\frac{k^{1/\alpha}}{c}$; hence, by condition (ii), $\pi(G) \geq k$. Otherwise, G has treewidth at most $\mathrm{poly}(k)$; a tree decomposition can be computed in time $\mathcal{O}(2^{\mathrm{poly}(k)}\mathrm{poly}(n))$ using the algorithm of Bodlaender [Bod96], and $\pi(G)$ can be computed using condition (iii) above. If the exponents are slightly increased, as specified in Theorem 7.34 (ii) and (iii), a perfect bramble can actually be constructed, and hence a witness can be provided. □

Using Lemma 7.33 one can see that our meta-theorem above can be applied to a variety of problems, such as vertex cover, edge dominating set (= minimum maximal matching), feedback vertex set, longest path, and maximum-leaf spanning tree. Whereas there already exist better FPT-algorithms for these problems, we do not know of a unifying argument like in Theorem 7.36 that provides singly-exponential FPT-algorithms for all these problems; also, this technique might be applicable to other problems, for which singly-exponential FPT-algorithms are not known yet. But the main significance of the theorem resides in the reasons discussed in the introduction of this work, regarding the graph minor theorem. Also, the algorithmic nature of Theorem 7.34 makes it possible to actually construct a *witness*, as specified by Theorem 7.36; this was, in general, not achieved by previous results.

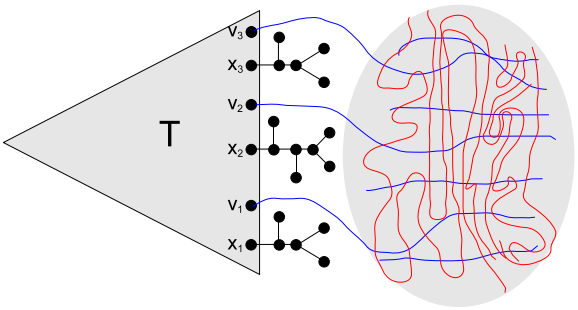


Figure 7.5: A labeled tree-ordered web encoding 010.

7.4 Labeled Tree-Ordered Webs

Our aim in this section is to show that any graph of large enough treewidth either contains a large clique minor or a grid-like minor attached to a special tree that can be completely defined and ordered by MSO_2 formulas; in both cases, we want the structure to be of order polynomial in the treewidth. Our basis is given by Theorem 7.27 (i) that provides us either with a large clique minor or a certain k-web of order 3 or 4 that contains a large grid-like minor. What we need to do is to consider one of the subcubic trees of the k-web and prune it appropriately, so that the tree and a linear order on its vertices become definable in MSO_2.

The structure we are after, which we call a *labeled tree-ordered web*, is indicated in Figure 7.5. The oval gray area schematically represents a grid-like minor; some of its paths are attached to the tree T. We would like to think of these paths as the first columns of the first row of the grid-like minor, ordered from left to right by the tree. In Chapter 8, we would like to label these columns by a binary word; we accomplish this by using what we call *single crosses* and *double crosses*. These are small subgraphs adjacent to some leaves of the tree T such that each path of the grid-like minor that starts at a leaf of T is preceded by a single cross, if it is to be labeled by 0, and by a double cross, if it is to be labeled by 1. In Figure 7.5, the paths of the grid-like minor starting in T are adjacent to the leaves v_1, v_2, and v_3 and the corresponding crosses are adjacent to the leaves x_1, x_2, and x_3, and encode the word 010.

It turns out to be more convenient to work with *walls* instead of grid-minors in MSO_2. Hence, we start by reviewing walls and then proceed to show step-by-step how to define and obtain a labeled tree-ordered web in graphs of large treewidth.

7.4.1 Topological Minors and Walls

We start by recalling some crucial definitions:

Definition 7.37. A *subdivision* of a graph H is a graph H' obtained from H by iteratively replacing some edges by paths of length 2. The original vertices of H in H' are called the *nails* of H in H'. If a graph G contains a subdivision of H as a subgraph, we call H a *topological minor* of G.

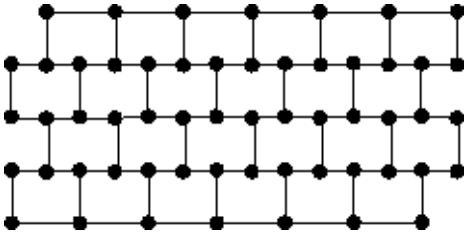


Figure 7.6: Elementary 4×6-wall.

If H is a topological minor of G, then H is also a minor of G. However, the reverse is not true in general *unless* H is of maximum degree 3; in this case, H is a minor of G if and only if it is a topological minor of G.

For two sets of disjoint paths $\mathcal{P}$ and $\mathcal{Q}$ in a graph G, we defined $(\mathcal{P}, \mathcal{Q})$ to be a *topological grid-like minor of order* ℓ if their intersection graph $\mathcal{I}(\mathcal{P}, \mathcal{Q})$ contains a subdivision of $\mathbf{K}_\ell$. We define the *nails* of $(\mathcal{P}, \mathcal{Q})$ to be those paths from $\mathcal{P} \cup \mathcal{Q}$ that are nails of the subdivision of $\mathbf{K}_\ell$ in $\mathcal{I}(\mathcal{P}, \mathcal{Q})$.

Definition 7.38. Let $n, m > 0$ be integers. An *elementary* $n \times m$*-wall* is a graph with vertex set $V := \{(1, 2j-1) : 1 \leq j \leq m+1\} \cup \{(i, j) : 1 < i \leq n, 1 \leq j \leq 2m+2\} \cup \{(n+1, 2j-t) : 1 \leq j \leq m+1 \text{ and } t := (n \bmod 2)\}$ and edge set

$$
\begin{aligned}
E := {} & \{(1, 2j-1), (1, 2(j+1)-1) : 1 \leq j \leq m\} \quad \text{// horizontal edges in row 1} \\
& \cup \{(n+1, 2j-t), (n+1, 2(j+1)-t) : 1 \leq j \leq m \text{ and } t := (n \bmod 2)\} \\
& \qquad\qquad \text{// horizontal edges in row } n+1 \\
& \cup \{(i, j), (i, j+1) : 2 \leq i \leq n, 1 \leq j < 2m+2\} \quad \text{// further horizontal edges} \\
& \cup \{(i, j), (i+1, j) : 1 \leq i \leq n, 1 \leq j \leq 2m+2, i \text{ and } j \text{ even }\} \quad \text{// vertical edges} \\
& \cup \{(i, j), (i+1, j) : 1 \leq i \leq n, 1 \leq j \leq 2m+2, i \text{ and } j \text{ odd }\}. \quad \text{// vertical edges}
\end{aligned}
$$

An $n \times m$*-wall* is a subdivision of an elementary $n \times m$-wall. The *nails* of a wall are the nails of the subdivision of the elementary wall it is obtained from. An elementary wall is a graph as illustrated in Figure 7.6.

We will always think of the vertices of a wall as being numbered in such a way that $(1, 1)$ is the vertex in the "bottom-left corner". The "bottom row" of an $n \times m$-wall is then the row 1. Note that since an elementary wall has maximum degree 3, any graph that has an elementary $n \times m$-wall as a minor contains an $n \times m$-wall as a subgraph. Furthermore, any $n \times m$-wall contains an $n \times m$-grid as a minor; conversely, any $n \times (2m+1)$-grid contains an elementary $n \times m$-wall as a subgraph.

7.4.2 Tree-Webs

The notion of a *tree-web*, defined below, is central to this part of our work; in the subsequent sections, we will gradually refine this notion until we finally obtain the structure that we need.

Definition 7.39. A *tree-web* of order ℓ is a tuple $\mathcal{W} = (G, T, r, A, \mathcal{P}, \mathcal{Q})$, so that

1. T is a subcubic tree rooted at r,
2. $(\mathcal{P}, \mathcal{Q})$ is a topological grid-like minor of order ℓ^2 whose nails are paths from $\mathcal{P}$,
3. G is a graph of maximum degree 4 with $T \cup \bigcup \mathcal{P} \cup \bigcup \mathcal{Q} \subseteq G$,
4. T only intersects with nails of $(\mathcal{P}, \mathcal{Q})$,
5. the paths from $\mathcal{P}$ that are nails are either disjoint from T or intersect T in exactly one endpoint, and
6. $A = V(T) \cap V(\bigcup \mathcal{P})$ is flat in T.

The vertices of A are called the *good* vertices of $\mathcal{W}$. The paths in $\mathcal{P}$ that start at a vertex in A are called *good* paths.

In case $G = T \cup \bigcup \mathcal{P} \cup \bigcup \mathcal{Q}$ and *all* the nails in $\mathcal{P}$ are good, and hence intersect T, i.e. $|A| = \ell^2$, we call the structure a *full tree-web*. A *sub-tree-web* of a tree-web $\mathcal{W} = (G, T, r, A, \mathcal{P}, \mathcal{Q})$ is a tree-web $\mathcal{W}' = (G', T', r', A', \mathcal{P}', \mathcal{Q}')$ with $G' \subseteq G$. In this case, we write $\mathcal{W}' \subseteq \mathcal{W}$. A *full subtree* of a rooted tree (T, r) is the connected component of $T - e$ not containing r, for some $e \in E(T)$.

Definition 7.40. A tree-web $\mathcal{W} = (G, T, r, A, \mathcal{P}, \mathcal{Q})$ is *nice* if

1. $T \cup \bigcup \mathcal{P} \cup \bigcup \mathcal{Q}$ has no vertex of degree 1 except maybe r,
2. if $P = v_0, \ldots, v_k$ is a good path with $v_0 \in A$, v_1 does not lie on any other path,
3. every full subtree of T with at least 2 vertices contains at least 2 good vertices.
4. every leaf of T is good, and
5. the neighbor of every leaf of T in T is good.

Note that the last two conditions are implied by the third. The proof of Lemma 7.42 below is based on the combinatorial Lemma 7.41.

Lemma 7.41. *Let $G := \mathbf{K}_k$ be a clique on k vertices and assume at most k edges of G are colored red and the rest are colored blue. Then G contains a blue clique H of size $\lfloor k/3 \rfloor$ that can be found in polynomial time.*

Proof. We prove our claim by induction on k. Let $k \geq 3$ and consider the following cases:

(i) Assume the red degree of every vertex is at most 1; then we can obviously take half the vertices into H.

(ii) Assume the red degree of every vertex is exactly 2. Then the red subgraph consists of a number of cycles. From each cycle of size t, we can include every second vertex in H, hence obtaining at least $\lfloor t/2 \rfloor \geq t/3$ vertices ($t \geq 3$).

(iii) Otherwise G has a vertex u of red degree at most 1 and a vertex v of red degree at least 2. If u has red degree 1, let w be its red neighbor, otherwise let w be an arbitrary vertex (note that it might be $v = w$). Define $D := \{u, v, w\}$, $d := |D|$, and $G' := G - D$. Note that d is 2 or 3 and G' is a clique on $k - d$ vertices having at most $k - d$ red edges. By the induction hypothesis, we can find a blue clique H' in G' of size at least $\lfloor (k - d)/3 \rfloor$. But since u has only blue edges to G', we can add u to H' to obtain H of size at least $\lfloor (k - d)/3 \rfloor + 1 \geq \lfloor k/3 \rfloor$. □

Lemma 7.42. *Given a (full) tree-web $\mathcal{W}$ of order ℓ, one can construct a nice (full) tree-web $\mathcal{W}'$ of order at least $\lfloor \ell/2 \rfloor$ with $\mathcal{W}' \subseteq \mathcal{W}$ in polynomial time.*

Proof. Vertices of degree 1 are irrelevant to a tree-web and can be always removed; hence, we obtain and can always maintain property (1) of a nice tree-web. If a full subtree contains only one good vertex, we can extend the good path starting at that vertex to the root of the subtree and remove the rest of the subtree. Thus, we can always guarantee property (3), which implies properties (4) and (5). None of these operations changes the order of the tree-web.

It remains to show property (2). If a good path $P = v_0, \ldots, v_k$ intersects another path, i.e a path $Q \in \mathcal{Q}$ at vertex v_1, we co lour Q red. Since the number of good paths is at most ℓ^2, we obtain at most ℓ^2 red paths in $\mathcal{Q}$. Now, consider the subdivision H of $\mathbf{K}_{\ell^2}$ that is contained in $\mathcal{I}(\mathcal{P}, \mathcal{Q})$; at most ℓ^2 of the paths in H that correspond to a subdivided edge of $\mathbf{K}_{\ell^2}$ contain a red vertex $Q \in \mathcal{Q}$; by Lemma 7.41, we can find a subdivision H' of $\mathbf{K}_{\lfloor \ell^2/3 \rfloor}$ in H that contains no red vertices and whose nails are a subset of the nails of H. Hence, by considering only the paths $\mathcal{P}' := \mathcal{P} \cap V(H')$ and $\mathcal{Q}' = \mathcal{Q} \cap V(H')$, we still have a subdivision of $\mathbf{K}_{\lfloor \ell^2/3 \rfloor}$ in $\mathcal{I}(\mathcal{P}', \mathcal{Q}')$. □

Using Theorem 7.27 (i) and Lemma 7.42, we can easily prove:

Lemma 7.43. *There is a constant c and a polynomial-time algorithm that given a graph G of treewidth at least $c\ell^{14}$ finds an $\ell \times \ell$-wall or a full nice tree-web of order ℓ in G.*

Proof. By choosing the constant c appropriately, theorem 7.27 (i) implies that G contains either $\mathbf{K}_{8\ell^2}$ as a minor or a topological grid-like minor of order $8\ell^2$. In the former case we are done, since an elementary $\ell \times \ell$-wall has maximum degree 3, is contained in $\mathbf{K}_{8\ell^2}$, and is thus a topological minor of G implying that G contains an $\ell \times \ell$-wall as a subgraph.

In the latter case, let $(T, (T_i)_{1 \leq i \leq t}, (A_i)_{1 \leq i \leq t}, B)$ be the k-web of order t ($t = 3$ or 4) and $(\mathcal{P}, \mathcal{Q})$ the topological grid-like minor returned by the algorithm. The theorem states that $\mathcal{P}$ is the set of paths connecting T_1 and T_2. Let N be the set of nails in the model of $\mathbf{K}_{8\ell^2}$ in $\mathcal{I}(\mathcal{P}, \mathcal{Q})$. W.l.o.g. we may assume that $\mathcal{P}$ contains at least half of N; hence, by just considering the nails in $\mathcal{P}$, we still have a subdivision of $\mathbf{K}_{4\ell^2}$. By deleting T_2,T_3, and T_4 (if existent), we almost obtain a full tree-web of order 2ℓ, except that the paths in $\mathcal{P}$ that are not nails also intersect with T_1; but we can delete the first edge of these paths and obtain the desired full tree-web. The root of the tree can be chosen arbitrarily. Finally, we obtain our claim by appealing to Lemma 7.42. □

The following operation is essential for the proofs that follow.

Definition 7.44. Given a tree-web $\mathcal{W} = (G, T, r, A, \mathcal{P}, \mathcal{Q})$ and a good vertex $v \in A$, starting a path $P = vv_1 \ldots v_k$ of $\mathcal{P}$, the operation $\textsc{Cut}(v)$ is defined as removing the edge v_1v_2 from G, adding the edge vv_1 to T, removing the vertex v from A, and iteratively removing vertices of degree 1 from P.

See Figure 7.7 for an illustration. Note that by starting with a nice tree-web, this operation does not affect the order of the tree-web. Next, we would like to identify a unique root for a tree-web:

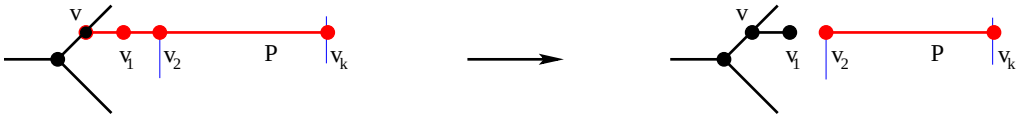


Figure 7.7: Illustration of the operation $\textsc{Cut}(v)$.

Definition 7.45. A tree-web $\mathcal{W} = (G, T, r, A, \mathcal{P}, \mathcal{Q})$ *admits a definable root* if it contains exactly one vertex $r \in V(T)$ with $\deg_T(r) = 1$ and $\deg_G(r) = 3$ such that two components of $G - r$ are single vertices s_1, s_2 and the third contains at least one edge.

Lemma 7.46. *Given a full nice tree-web $\mathcal{W} = (G, T, r, A, \mathcal{P}, \mathcal{Q})$ with at least 3 good vertices, one can construct a sub-tree-web $\mathcal{W}' = (G', T', r', A', \mathcal{P}', \mathcal{Q}')$ of the same order in polynomial time such that $\mathcal{W}'$ is nice, admits a definable root, and $|A'| \geq |A|/3$.*

Proof. If T has a vertex v of degree 3, one of the components of $T - v$ contains at least $1/3$ of the good vertices; we prune the other two to become a single vertex each to obtain $\mathcal{W}'$ with root v. If, on the other hand, T is a path, one of its endpoint v_1 is a good vertex connecting a path P_1 and its neighbor v_2 is a good vertex connecting a path P_2. By deleting the first edge of P_1 and applying the operation $\textsc{Cut}(v_2)$, v_2 becomes a definable root while losing only 2 good vertices. Since $\mathcal{W}$ is nice, these operations do not change the order of the tree-web; and by deleting redundant vertices of degree 1, we can make sure that $\mathcal{W}'$ is nice, too. □

7.4.3 Trees admitting a definable ordering

In this section we show how to prune a given rooted tree T with maximum degree 3, so that there is an MSO_2-formula (not depending on T) which at each branching node of the tree distinguishes between the left and the right subtree. Assume we are given a subcubic tree T with a root r and a set X of vertices of the tree marked as *good* and we would like to retain as many good vertices as possible. Throughout this section, X will always denote the set of good vertices; and we assume $\deg_T(r) = 1$. We use the following notation (see Figure 7.8 for illustrations):

- If $v \in V(T)$ then the children of v are all neighbors of v not on the unique path from v to r.
- A *leaf* of T is a node of degree 1 in T, except r. A *good leaf* is a leaf that is good.
- A vertex is called *leafy* if it has degree 3 and is adjacent to a leaf.
- A *branching vertex* of T is a vertex of degree 3 in T. A *proper branching vertex* is a branching vertex that is not leafy.
- An *artificial leaf* is a leaf that is not good and is adjacent to a branching vertex.
- Let $v \in V(T)$ be a vertex with child $u \in V(T)$ and $e = \{v, u\}$. The subtree T_u of T *rooted at* u is the component of $T - e$ containing u. The *extended subtree* of u is defined as $T_u \cup e$.
- $\textsc{subtree}_i(v)$ denotes the extended subtree of the i^{th} child of v, where we number the children arbitrarily.

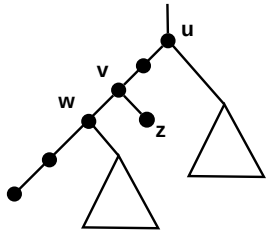


Figure 7.8: The vertices u, v, and w are properly marked; v is leafy and z an artificial leaf; u and w are proper branching vertices.

- $\text{CBV}(T, v)$: closest branching vertex to v in T_v; or the leaf of T_v if T_v is a path; is defined only if v has degree 1 in T_v.
- $g_X(T) := |X \cap V(T)|$ is the number of good vertices in T. We omit the index $\cdot_X$ if it is clear from context.

Definition 7.47. Let (T, r) be a rooted subcubic tree;

- two vertices $u, v \in V(T)$ are *topological neighbors* if they are linked by a path whose inner vertices all have degree 2 in T;
- a branching vertex v is called *properly marked* if it has a leaf or a leafy vertex as a topological neighbor in the subtree rooted at v; and
- T is called *properly marked* if every branching vertex of T is properly marked.

We now define a pruning algorithm $\text{PRUNE}(T, r)$ which, given a rooted subcubic tree (T, r) outputs a tree (T', r) that is properly marked (see Figure 7.9).

Algorithm $\text{PRUNE}(T, r)$**.**
Input. subcubic rooted tree (T, r) with $\deg_T(r) = 1$.
Output. a properly marked subcubic rooted tree (T', r) with $T' \subseteq T$.

If T is a simple path than return T. Otherwise, let $v := \text{CBV}(T, r)$, R be the path from r to v, $T_1 := \text{SUBTREE}_1(v)$, and $T_2 := \text{SUBTREE}_2(v)$ with $g(T_1) \leq g(T_2)$.

1. If one of T_1, T_2 is a path, say T_i, return the tree obtained from T by replacing T_{3-i} by $\text{PRUNE}(T_{3-i})$.
2. Otherwise, let $u_1 := \text{CBV}(T_1, v)$. Let $T_{11} := \text{SUBTREE}_1(T_1, u_1)$ and $T_{12} := \text{SUBTREE}_2(T_1, u_1)$ with $g(T_{11}) \leq g(T_{12})$. Let T_1' be the tree obtained from T_1 by cutting T_{11} down to a single edge and replacing T_{12} by $T_{12}' := \text{PRUNE}(T_{12}, u_1)$. Finally, return T' as the union of R, T_1', and $T_2' := \text{PRUNE}(T_2)$.

Lemma 7.48. *Let (T, r) be a rooted subcubic tree and $X \subseteq V(T)$. T contains a properly marked subtree T' such that $g_X(T') \geq g_X(T)^{\frac{2}{3}}$. Furthermore, T' can be computed in polynomial time on input (T, r).*

Proof. Let $T' := \textsc{Prune}(T, r)$. We claim that (T', r) fulfills the requirements of the lemma. We prove the claim by induction on the order $n := |T|$ of T. If T is a path there is nothing to show. Otherwise, the fact that T' is properly marked is immediate from our recursive construction by induction. It remains to bound the number of good vertices that remain after the pruning. We first observe that for all $\frac{1}{2} \leq \beta \leq 1$

$$\left(\frac{1-\beta}{2}\right)^{\frac{2}{3}} + \beta^{\frac{2}{3}} \geq 1\,. \tag{7.1}$$

If q, q_1, q_2 are non-negative integers with $q = q_1 + q_2$ and $q_1 \leq q_2$, we have $q_2 = \beta q$ and $q_1 = (1-\beta)q$, for some $\beta \geq \frac{1}{2}$. Hence, we obtain with Inequality (7.1)

$$\begin{aligned} q_1^{\frac{2}{3}} + q_2 \geq q_1 + q_2^{\frac{2}{3}} \geq q_1^{\frac{2}{3}} + q_2^{\frac{2}{3}} &\geq (\frac{q_1}{2})^{\frac{2}{3}} + q_2^{\frac{2}{3}} \\ &= q^{\frac{2}{3}} \cdot ((\frac{1-\beta}{2})^{\frac{2}{3}} + \beta^{\frac{2}{3}}) \\ &\geq q^{\frac{2}{3}} = (q_1+q_2)^{\frac{2}{3}}\,. \end{aligned} \tag{7.2}$$

Let v, R. T_1, and T_2 be defined as in the algorithm. Define $q_0 := g(R-v)$, $q_1 := g(T_1-v)$, $q_2 := g(T_2)$, $q := q_1 + q_2$, and $q' := g(T'_v)$. First, note that it suffices to show $q' \geq q^{\frac{2}{3}}$ since this implies

$$g(T') = q_0 + q' \geq q_0 + q^{\frac{2}{3}} \geq (q_0+q)^{\frac{2}{3}} = g(T)^{\frac{2}{3}}$$

by Inequality (7.2). Consider the following cases:

(i) If T_1 is a path, then $q' \geq q_1 + q_2^{\frac{2}{3}} \geq q^{\frac{2}{3}}$ by Inequality (7.2). Similarly, if T_2 is a path, then $q' \geq q_1^{\frac{2}{3}} + q_2 \geq q^{\frac{2}{3}}$.

(ii) Otherwise, let T'_{12} and T'_2 be defined as in Step 2 of the algorithm and let $q'_2 := g(T'_2)$ and $q'_{12} := g(T'_{12})$. Furthermore, let P be the path from u to v excluding u and v and let $q_P := g(P)$. Using Inequality (7.2) twice more, we obtain

$$\begin{aligned} q' = q_P + q'_{12} + q'_2 &\geq q_P + \left(\frac{q_1 - q_P}{2}\right)^{\frac{2}{3}} + q_2^{\frac{2}{3}} \\ &\geq \left(\frac{q_1}{2}\right)^{\frac{2}{3}} + q_2^{\frac{2}{3}} \geq q^{\frac{2}{3}}\,. \end{aligned}$$

□

Once we have a properly marked tree, it is possible to identify left and right subtrees in a proper way using parity considerations, as follows.

Definition 7.49. Let (T, r) be a subcubic tree rooted at vertex r of degree 1 and X a set of good vertices that lies flat in T. We say the tuple (T, r, X) *admits a definable order* if all leaves are artificial or good and for all branching vertices v with extended

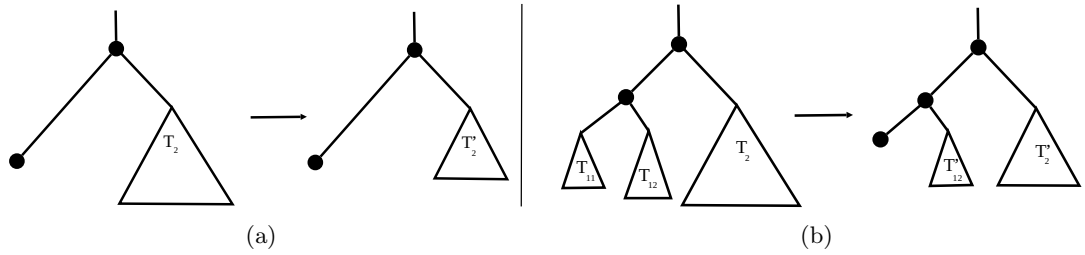


Figure 7.9: (a) Case (1) of algorithm PRUNE; (b) case (2) of algorithm PRUNE.

subtrees $T_i := \text{SUBTREE}_i(v)$, for $i = 1, 2$, exactly one of the following is true. Along with the following conditions we will label some subtrees as *left* and others as *right*.

1. At least one of T_1, T_2 is a single edge, say T_1. If T_2 is also a single edge, then exactly one of T_1 and T_2 contains a good leaf, say T_1; in either case, T_1 is left and T_2 is right.
2. Exactly one of T_1, T_2 is a simple path, say T_1. Then T_1 is left and T_2 is right.
3. Let u_i be the closest proper branching vertex to v in T_i if one exists; otherwise let u_i be the good leaf of T_i farthest away from v. Let P_i the path connecting v and u_i in T_i. We define g_i to be the number of vertices on $P_i - v$ that are *good or leafy*. We require that exactly one of g_1, g_2 is odd, say g_1; then T_1 is left and T_2 is right.

The *canonical order* $\leq_T$ of (T, r) is defined as follows. Let $x \neq y \in V(T)$ and let v be the closest common ancestor of x, y. Then $x \leq_T y$ if and only if $v = x$ or x is in the left subtree of v and y in the right.

Lemma 7.50. *Let (T, r) be a subcubic tree rooted at vertex r of degree 1 and $X \subseteq V(T)$ a given set of good vertices that lies flat in T. T contains a subtree T' and a set $X' \subseteq X \cap T'$ with $|X'| \geq |X|^{\frac{2}{3}}/2$ such that (T', r, X') admits a definable order and X' is* totally *ordered by the canonical order $\leq_{T'}$. Furthermore, T' can be computed in polynomial time.*

Proof. W.l.o.g. we assume all the leaves of T are good; otherwise we go from T to the smallest subtree of T containing the root and all good vertices; hence, all the leaves of T are good leaves. Then we apply Lemma 7.48 to obtain a properly marked subtree T'' of T and a set $X'' := X \cap T''$ with $|X''| \geq |X|^{\frac{2}{3}}$. Note that when counting the number of good vertices of T'' in Lemma 7.48, we do not consider the leaf that replaces a subtree in step (2) of algorithm PRUNE a good vertex, even though it might happen to be one; hence, we consider all these leaves artificial leaves and are free to remove them without losing good vertices. All other leaves are still good vertices. Now we consider each branching vertex v of T'' in a bottom-up fashion, i.e. in a post-order traversal of the tree, and consider the following cases; let T_i,u_i, and g_i be defined as in Definition 7.49:

(i) If both of T_1, T_2 are single edges, ignore v.
(ii) Suppose T_1 and T_2 are both simple paths of length at least 2 and g_1 and g_2 have the same parity. Then u_1 and u_2 are both good leaves. If there is no other good vertex

in T_1, we cut T_1 down to a single edge, i.e. make it an artificial leaf. Otherwise, let w be the good vertex closest to u_1 and replace T_1 by the path from v to w. In either case, we lose exactly one good vertex in the subtree rooted at v; and our construction ensures that case (ii) does not occur for any ancestor of v in the tree.

(iii) Otherwise, if at least one of T_1, T_2 is a simple path, ignore v.

(iv) Otherwise v is a proper branching vertex and Lemma 7.48 guarantees that v has a leafy vertex, adjacent to an artificial leaf w, as a topological neighbor in one of its subtrees. If g_1 and g_2 are of the same parity, we simply remove w and obtain our desired property without losing any good vertex.

We let T' be the tree obtained after the traversal above is finished and $X' := T' \cap X - D$, where D contains one of every two good leaves that are siblings as in case (i). Then, it is evident by our construction that (T', r, X') admits a definable order and all leaves are either good or artificial. Furthermore, X' contains at least half the vertices of X'', since the subtrees on which case (i) or (ii) apply are all disjoint and at most half of their good vertices are not included in X'. □

7.4.4 Tree-Ordered Webs

We show how to prune the tree T of a given nice full tree-web $(G, T, r, A, \mathcal{P}, \mathcal{Q})$, so that there is an MSO_2-formula which can detect the nodes of T in G and at each branching node of the tree distinguishes between the left and right subtree.

Definition 7.51. Let $\mathcal{W} = (G, T, r, A, \mathcal{P}, \mathcal{Q})$ be a tree-web:

- a *leaf-mark* of $\mathcal{W}$ is a path $v_1v_2v_3$ in G such that v_1 is a good leaf of T, v_2 is of degree 2 in G and v_3 is of degree 1 in G (see Figure 7.10);
- we use the notions *topological neighbor* and *properly marked* with respect to *degrees in* G (as opposed to degrees in T in the previous subsection; this does make an important difference, as good vertices have degree 2 in T but degree 3 in G);
- a vertex $v \in V(G)$ is *special in* G if it has degree 3 or 4 and is *not* properly marked, i.e. does not have a leaf or a leafy vertex as a topological neighbor;
- we let *spec*(G) denote the set of special vertices in G.

Definition 7.52. A *tree-ordered web* of order ℓ is a tuple $(G, T, r, A, \mathcal{P}, \mathcal{Q})$ such that

1. $(G, T, r, A, \mathcal{P}, \mathcal{Q})$ is a tree-web of order ℓ admitting the definable root r;
2. A is the set of vertices of degree 3 in G not in *spec*(G) but having a topological neighbor in *spec*(G);

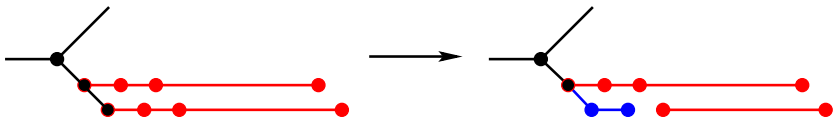


Figure 7.10: Turning a good leaf with a good neighbor into a leaf with a leaf-mark.

3. T is contained in the component of $G - \mathit{spec}(G)$ containing r;
4. every leaf of T is either artificial or incident to a leaf-mark;
5. (T, r, A) admits a definable order;
6. G consists only of $T \cup \bigcup \mathcal{P} \cup \bigcup \mathcal{Q}$, the marking of r, and the leaf-marks; and
7. no vertex of $V(\bigcup \mathcal{P}) \cup V(\bigcup \mathcal{Q})$ has degree 1 in G.

A main ingredient of the proof of Lemma 7.53 below is the observation that Definition 7.49 allows us to cut a good path and make it an artificial leaf without destroying the definable order; this is because in the third case of Definition 7.49, we consider vertices that are *good or leafy* and the operation Cut only turns a good vertex into a leafy vertex. Hence, we can cut away about every second good path to ensure that vertices of the tree do not land in $\mathit{spec}(G)$. We also observe that if the number of the leaves of the tree is large enough, we can just keep the good paths starting at the leaves; and otherwise, the number of proper branching vertices is small and we do not need to cut away too many good paths.

Lemma 7.53. *There exists a constant c such that if $\mathcal{W}_0 = (G_0, T_0, r_0, A_0, \mathcal{P}_0, \mathcal{Q}_0)$ is a given nice full tree-web of order $c\ell$, then there exists a tree-web $\mathcal{W} = (G, T, r, A, \mathcal{P}, \mathcal{Q})$ with $\mathcal{W} \subseteq \mathcal{W}_0$ and a vertex $r \in V(G)$ such that $(G, T, r, A, \mathcal{P}, \mathcal{Q})$ is a tree-ordered web of order ℓ with $|A| \geq 15\ell$; furthermore, $\mathcal{W}$ can be computed in polynomial time.*

Proof. First, we apply Lemma 7.46 to obtain a nice tree-web $\mathcal{W}_1 = (G_1, T_1, r, A_1, \mathcal{P}_1, \mathcal{Q}_1)$ with definable root r. The lemma guarantees that $|A_1| \geq |A_0|/3$. Recall that this way, r is of degree 1 in T but is never considered a leaf.

Since $\mathcal{W}_1$ is nice, every leaf of T_1 is good and is adjacent to another good vertex of T_1. Let $T_1' \subseteq T_1$ be the subtree of T_1 obtained by removing all the leaves of T_1. Note that all the leaves of T_1' are still good. Let $A_1' = A_1 \cap V(T_1')$ and observe that $|A_1'| \geq |A_1|/2$. Next, we invoke Lemma 7.50 on (T_1', r) and A_1' to obtain a tuple (T_2, r, A_2) admitting a definable order with $|A_2| \geq |A_1'|^{\frac{2}{3}}/2$. The lemma guarantees that every leaf of T_2 is either artificial or good.

Let v be a good leaf of T_2. By our construction above, there must exist an edge $vu \in E(T_1) - E(T_2)$. Now (i) if u is not a good vertex of T_1, then u cannot be a leaf of T_1, and hence there must exist another edge $uw \in E(T_1) - E(T_2)$ with $w \neq v$; in this case, vuw is a leaf-mark for v; (ii) otherwise, u is a good vertex of T_1 starting a path $P = uw_1w_2\ldots$; we delete the edge w_1w_2 as in the Cut operation to obtain the leaf-mark vuw_1 for v (see Figure 7.10); since we started with a nice tree-web, this operation does not change the order of the tree-web.

We apply the procedure above to every good leaf of T_2, remove all edges of T_1 from G that do not appear in T_2 or in leaf-marks, and iteratively remove redundant vertices of degree 1 appearing in $(\mathcal{P}, \mathcal{Q})$. Let $\mathcal{W}_2 = (G_2, T_2, r, A_2, \mathcal{P}_2, \mathcal{Q}_2)$ be the resulting tree-web.

Let b be the number of proper branching vertices of T_2 and t the number of good leaves. Since T_2 is subcubic, we have $b \leq t$. We obtain the tree-web $\mathcal{W}_3 = (G_3, T_3, r, A_3, \mathcal{P}_3, \mathcal{Q}_3)$ as follows:

(i) If $t \geq |A_2|/4$, we let A_3 be the set of good leaves of T_2 and obtain $\mathcal{W}_3$ by performing the operation Cut on every good vertex not in A_3.

(ii) Otherwise, consider each proper branching vertex v of T_2 and let P_1, P_2 be the paths from v to the closest proper branching vertex or good leaf of each of the two extended subtrees of v. Let $u_1, \ldots, u_p$ be the good vertices on P_1, in this order, and similarly, $w_1, \ldots, w_q$ the good vertices on P_2. If one of p, q is 1, we assume w.l.o.g. that $p = 1$; if both are 1, then we let $w_1, \ldots, w_q$ belong to the subtree that contains an artificial leaf (note that there is such a subtree, since the parities of the total number of good or leafy vertices must be different). Apply the operation $\textsc{Cut}(u_i)$ for every odd $1 \leq i \leq p$ and the operation $\textsc{Cut}(w_i)$ for every even $1 \leq j \leq q$. This way, it is guaranteed that v retains a leafy vertex as a topological neighbor and still, at least $\lfloor (p+q)/2 \rfloor$ good vertices on $P_1 \cup P_2$ are left. If $p + q$ is odd, we charge one unit of penalty to v. Let T_3 and A_3 be the tree and good vertices after this operation is performed on every proper branching vertex. Since every proper branching vertex is charged to at most once, the number of good vertices that remain is at least $|A_2|/2 - b$; but we have $b \leq t \leq |A_2|/4$, and hence we obtain $|A_3| \geq |A_2|/4$.

We claim that $\mathcal{W} := \mathcal{W}_3$ is the desired tree-ordered web specified in the lemma. Indeed, note that since we started with a nice tree-web, none of the operations above changed the order of the tree-webs we worked with; also every branching vertex and every good vertex in T is properly marked by a leafy topological neighbor while vertices of $\mathcal{P} \cup \mathcal{Q}$ do not have this property and thus belong to $spec(G)$; furthermore, (T_3, r, A_3) admits the same definable order as (T_2, r, A_2) because the $\textsc{Cut}$ operation only changes good vertices into leafy vertices, which does not make a difference in the definable order. Hence, all the properties of Definition 7.52 are fulfilled. Finally, recall that $|A_0| = c^2\ell^2$; we have $|A_3| \geq \frac{|A_0|^{\frac{2}{3}}}{27} = \frac{c^{\frac{4}{3}}}{27} \cdot \ell^{\frac{4}{3}}$, and so $|A_3| \geq 15\ell$ is also fulfilled if the constant c is large enough (if ℓ is larger than a constant, then $c = 1$; otherwise $c \leq 91$ suffices). □

7.4.5 Labeling Tree-Ordered Webs

We will show next how to encode a word $w := w_1 \ldots w_t \in \{0,1\}^\star$ in a tree-ordered web of order $2t$. We first need the following simple combinatorial lemma.

Lemma 7.54. *Let G be a directed graph on k vertices with maximum outdegree d. Then G contains an independent set of size $\left\lceil \frac{k}{2d+1} \right\rceil$ which can be computed in polynomial time.*

Proof. As the maximal outdegree of each vertex is at most d, the graph contains at most kd edges, i.e. in the underlying undirected graph, the sum of the vertex degrees is at

Figure 7.11: A (a) single and a (b) double cross.

most $2kd$. Hence, there is a vertex of total degree at most $2d$. We can add it to the independent set and remove all its in- and out-neighbors. Proceeding in this way we find an independent set of size $\left\lceil \frac{k}{2d+1} \right\rceil$. □

A *single cross* is a subcubic tree with four leaves having the shape depicted in Figure 7.11 (a); a *double cross* is a subcubic tree with five leaves having the shape depicted in Figure 7.11 (b) (where the dashed lines indicate paths). The right-most vertex of each cross, as drawn in Figure 7.11, is called the *base* of the cross.

Definition 7.55. A *labeled tree-ordered web of order ℓ and length k* is a tuple $\mathcal{W} := (G, T, r, A, \mathcal{P}, \mathcal{Q}, X, C)$ where

1. $(G[V(G - C) \cup X], T, r, A, \mathcal{P}, \mathcal{Q})$ is a tree-ordered web of order ℓ except that we require $(T, r, A \cup X)$ to admit a definable order instead of (T, r, A),
2. the root r does not have a leafy vertex as a topological neighbor,
3. C is a set of disjoint single and double crosses,
4. $X = V(T) \cap V(C)$ is the set of bases of the crosses in C and lies flat in T,
5. $|X| = |A| = k$,
6. if $X = \{x_1, \ldots, x_k\}$ and $A = \{v_1, \ldots, v_k\}$ then $x_1 \leq_T v_1 \leq_T x_2 \cdots \leq_T x_k \leq_T v_k$.

The word *encoded by* $\mathcal{W}$ is $w := w_1 \ldots w_k \in \{0,1\}^k$ with $w_i := 0$ if x_i is the base of a single cross in C and $w_i := 1$ if x_i is the base of a double cross of C. $\mathcal{W}$ is called *configurable* if C consists only of double crosses.

A labeled tree-ordered web encoding the word 010 is indicated in Figure 7.5.

Lemma 7.56. *For $\ell \geq 3$, let $\mathcal{W} = (G, T, r, A, \mathcal{P}, \mathcal{Q})$ be a given tree-ordered web of order 2ℓ with $|A| \geq 30\ell$. There exists a configurable labeled tree-ordered web $\mathcal{W}' = (G', T', r', A', \mathcal{P}', \mathcal{Q}', X', C')$ of order ℓ and length ℓ with $G' \subseteq G$ that can be computed in polynomial time.*

Proof. First, note that any good path $P \in \mathcal{P}$ can be easily transformed to a double cross: since P is a nail of the grid-like minor $(\mathcal{P}, \mathcal{Q})$ of order $4\ell^2$, P intersects with at least 3 paths of $\mathcal{Q}$ if $\ell \geq 1$. The first 3 of these paths can be cut in a way to create one of the double crosses depicted in Fig. 7.11 (b). By doing so, we could destroy at most 7 other paths: 3 paths of $\mathcal{Q}$ and 4 paths of $\mathcal{P}$ that might have intersected the 4 leaves of the double cross (the base is part of the tree T). Each path $R \in \mathcal{P} \cup \mathcal{Q}$ that is not a nail might be used on at most one subdivided edge connecting the nails $P_1, P_2 \in \mathcal{P}$ in the image of $\mathbf{K}_{4\ell^2}$ in $\mathcal{I}(\mathcal{P}, \mathcal{Q})$. Assign R arbitrarily, say, to P_1. If R is destroyed by building a cross, we consider P_1 being destroyed, instead.

Consider a digraph $\mathcal{D}$ having a vertex u_P for each good path in $\mathcal{P}$ and a directed edge from u_P to $u_{P'}$ if turning P into a double cross destroys P'. The maximum outdegree of this digraph is 7 and hence, by Lemma 7.54, there exists a set $Y_0 \subseteq A$ of size at least $\frac{|A|}{15} \geq 2\ell$ of good vertices such that the vertices in $\mathcal{D}$ that correspond to the good paths starting at Y_0 form an independent set in $\mathcal{D}$.

Let $Y := \{y_1, \ldots, y_{2\ell}\}$ be a subset of exactly 2ℓ vertices of Y_0 such that $y_1 \leq_T y_2 \leq_T \cdots \leq_T y_{2\ell}$. We define $X' := \{y_i \mid i \text{ is odd}\}$ and $A' := \{y_i \mid i \text{ is even}\}$. We transform every

good path starting at a vertex in X' into a double cross and remove all the paths that get destroyed. Let C' be the set of double crosses we obtain this way. We also perform the operation $\textsc{Cut}(v)$ on every good vertex that is not in Y. If the root r has a leafy vertex with leaf w as a topological neighbor, we remove w and repeat this process, if necessary; note that such leaves are irrelevant to the tree-order and can be safely removed. Finally, we repeatedly remove all redundant vertices of degree 1 of $(\mathcal{P}, \mathcal{Q})$.

Let $\mathcal{P}'$ and $\mathcal{Q}'$ be the (parts of) the paths that remain, T' the tree obtained from T after the cut operations, $r' := r$, and G' be the union of $T' \cup C' \cup \mathcal{P}' \cup \mathcal{Q}'$ with the marking of the root and the leaf-marks. We claim $\mathcal{W}' = (G', T', r', A', \mathcal{P}', \mathcal{Q}', X', C')$ is the desired configurable labeled tree-ordered web of order ℓ and length ℓ.

The fact that $(T', r', A' \cup X')$ admits a definable order follows on one hand, from the observation that Definition 7.49 allows turning a good vertex into a leafy vertex by the $\textsc{Cut}$ operation without changing the canonical order of the tree; and on the other hand, from the fact that the good vertices that started good paths that are now turned into crosses are now in X and thus still count as good vertices. Hence, the canonical order of $(T', r', A' \cup X')$ is indeed the same as the canonical order of (T, r, A). The number of destroyed paths is at most $8|X'| = 8\ell$, i.e. we lose at most 8ℓ nails of the subdivision of $\mathbf{K}_{4\ell^2}$ in $\mathcal{I}(\mathcal{P}, \mathcal{Q})$; hence $\mathcal{I}(\mathcal{P}', \mathcal{Q}')$ still contains a subdivision of $\mathbf{K}_{\ell^2}$ if $\ell \geq 3$. All other requirements of Definition 7.55 are immediate from our construction. □

The definition below is needed in Chapter 8:

Definition 7.57. If $\mathcal{W}$ is a labeled tree-ordered web of order ℓ^d and length ℓ encoding a word $w = w_1 \dots w_\ell$, we say that $\mathcal{W}$ *encodes* w *with power* d.

Theorem 7.58 sums up the main result of this section and, indeed, of this lengthy chapter:

Theorem 7.58. *Let a word $w = w_1 \dots w_\ell \in \{0,1\}^\star$, a graph G, and an integer d be given. There is a constant c such that if the treewidth of G is at least $c\ell^{14d}$ then G contains either an $\ell^d \times \ell^d$-wall or a labeled tree-ordered web $\mathcal{W}$ that encodes w with power d. Furthermore, either outcome can be computed in polynomial time.*

Proof. By applying Lemma 7.43 to G, we obtain either an $\ell^d \times \ell^d$-wall or a full nice tree-web of order $c'\ell^d$, for a suitable constant c' if c is chosen appropriately. In the former case, we are done and in the latter case, we invoke Lemma 7.53 and Lemma 7.56 in order and obtain a configurable labeled tree-ordered web $\mathcal{W}' = (G', T', r', A', \mathcal{P}', \mathcal{Q}', X', C')$ of order ℓ^d and length ℓ^d. We apply the operation $\textsc{Cut}(v)$ to all but the first ℓ good paths in A', remove all but the first ℓ double crosses in C, and cut some double crosses to single crosses according to w. The labeled tree-ordered web $\mathcal{W}$ that remains fulfills our requirements. Since all the Lemmas that we used require only polynomial time, the whole procedure does, too. □

8 Parameterized Intractability of Monadic Second-Order Logic[1]

In 1990, Courcelle proved a fundamental result stating that every property of graphs definable in *monadic second-order logic with edge set quantification* (MSO_2), the extension of first-order logic by quantification over sets of vertices and edges, can be decided in linear time on any class $\mathcal{C}$ of graphs of bounded treewidth. This theorem has important consequences both in logic and in algorithm theory. In the theory of efficient algorithms on graphs, it can often be used as a simple way of establishing that a property can be solved in linear time on graph classes of bounded treewidth. Besides being of interest for specific algorithmic problems, results such as Courcelle's and similar *algorithmic meta-theorems* lead to a better understanding how far certain algorithmic techniques, such as dynamic programming on bounded treewidth graphs, range; and also establish general upper bounds for the parameterized complexity of a wide range of problems. See [Gro07a, Kre09a] for recent surveys on algorithmic meta-theorems.

From a logical perspective, Courcelle's theorem establishes a sufficient condition for tractability of MSO_2 formula evaluation on classes of graphs or structures: whatever the class $\mathcal{C}$ may look like, if it has bounded treewidth, then MSO_2-model checking is tractable on $\mathcal{C}$. An obvious question is how tight the theorem actually is, i.e. whether it can be extended to classes of unbounded treewidth and if so, how large the treewidth of graphs in the class can be in general. Given the considerable interest in Courcelle's theorem, and the far-reaching consequences that extensions of this result to interesting classes of graphs of unbounded treewidth would have, it is surprising that not much is known about such limits for MSO_2-model checking. To fully understand the (parameterized) complexity of monadic second-order logic with respect to particular classes of graphs, we need to understand necessary conditions for tractability as much as sufficient conditions; but for some reason necessary conditions have so far not been studied in much depth.

In order to formally state and further discuss our results, also in relation to previous work, we need the following notion; it basically states that a class of graphs of unbounded treewidth actually contains sufficiently many graphs *witnessing* the large treewidth of the class and that these witnesses can be constructed efficiently.

[1] This chapter is based on joint work with Stephan Kreutzer [KT10a].

Definition 8.1. The treewidth of a class $\mathcal{C}$ of graphs is *strongly unbounded* by a function $f : \mathbb{N} \to \mathbb{N}$ if there is $\varepsilon < 1$ and a polynomial $p(x)$ such that for all $n \in \mathbb{N}$ there is a graph $G_n \in \mathcal{C}$ with

1. the treewidth of G_n is between n and $p(n)$ and is not bounded by $f(|G_n|)$ and
2. given n, G_n can be constructed in time 2^{n^ε}.

The degree of the polynomial p is called the *gap-degree* of $\mathcal{C}$ (with respect to f). The treewidth of $\mathcal{C}$ is *strongly unbounded polylogarithmically* if it is strongly unbounded by $\log^c n$, for all $c \geq 1$.

A first lower bound for the complexity of monadic second-order logic was recently given by [Kre09b]. He showed that under the exponential time hypothesis MSO_2-model-checking is not fixed-parameter tractable on any class of colored graphs where

(i) the treewidth is strongly unbounded by $\log^c n$, for a suitable constant c;
(ii) which are *closed under recolorings* for a fixed set Γ of colors, i.e. if $G \in \mathcal{C}$ and G' is obtained from G by coloring some vertices or edges with colors from Γ, then $G' \in \mathcal{C}$; and
(iii) where a grid-like minor of order polynomial in the treewidth can be constructed in polynomial time.

This paper establishes powerful logical tools for proving such intractability results and we will resort to some of these tools below. However, condition (iii) seems quite unnatural and makes the result somewhat artificial. But as we saw in the previous chapter, we proved in Theorem 7.27 that condition (iii) is fulfilled for *all* graphs! Hence, with our work in Chapter 7, we immediately obtain the following much more natural result:

Theorem 8.2. *Let $\mathcal{C}$ be a class of colored graphs closed under recoloring from a fixed set of colors Γ.*

(i) If the treewidth of $\mathcal{C}$ is strongly unbounded by $\log^{28^\gamma} n$, where $\gamma > 1$ is larger than the gap-degree of $\mathcal{C}$, then $\mathrm{MC}(\mathrm{MSO}_2, \mathcal{C})$ is not in XP, and hence not fixed-parameter tractable, unless Sat *can be solved in subexponential time.*

(ii) If the treewidth of $\mathcal{C}$ is strongly unbounded polylogarithmically then $\mathrm{MC}(\mathrm{MSO}_2, \mathcal{C})$ is not in XP unless all problems in the polynomial-time hierarchy can be solved in subexponential time.

Still, closure under colorings is from a logical perspective quite a strong condition as it allows to "mark" bad substructures in a graph. In this chapter we aim for an even stronger intractability result for MSO_2:

Theorem 8.3. *Let $\mathcal{C}$ be a class of graphs closed under subgraphs, i.e. $G \in \mathcal{C}$ and $H \subseteq G$ implies $H \in \mathcal{C}$.*

1. If the treewidth of $\mathcal{C}$ is strongly unbounded by $\log^{28^\gamma} n$, where $\gamma > 1$ is larger than the gap-degree of $\mathcal{C}$, then $\mathrm{MC}(\mathrm{MSO}_2, \mathcal{C})$ is not in XP, and hence not fixed-parameter tractable, unless Sat *can be solved in subexponential time.*

2. *If the treewidth of $\mathcal{C}$ is strongly unbounded polylogarithmically then* $\mathrm{MC}(\mathrm{MSO}_2, \mathcal{C})$ *is not in* XP *unless all problems in the polynomial-time hierarchy can be solved in subexponential time.*

Recall that $\mathrm{MC}(\mathrm{MSO}_2, \mathcal{C})$ refers to the parameterized model-checking problem for MSO_2 as defined in Section A.4. We will give a justification for the two conditions in Definition 8.1 below, once we have discussed some more related work. To give an example, the theorem implies that the class $\mathcal{C}$ of all (or all planar, bipartite, etc.) graphs G of treewidth $\mathrm{tw}(G) \leq \log^{29} |G|$ does not have fixed-parameter tractable MSO_2 model-checking unless SAT can be solved in subexponential time.

Related Work

Theorem 8.3 complements the intractability result of Theorem 8.2 and [Kre09b, KT10b] in that it refers to classes of graphs closed under subgraphs and does not require any colors, a much more natural condition.

In [Gro07a, Conjecture 8.3], Grohe conjectures [2] the following.

Conjecture 8.4 (Grohe [Gro07a])**.** *Let $\mathcal{C}$ be a class of graphs that is closed under taking subgraphs. Suppose that the treewidth of $\mathcal{C}$ is not polylogarithmically bounded, that is, there is no constant c such that* $\mathrm{tw}(G) \leq \log^c |G|$ *for every $G \in \mathcal{C}$. Then the model-checking problem of* MSO_2 *is not fixed parameter tractable on $\mathcal{C}$.*

Clearly, with current technology there is no hope to prove any such conjecture without relating it to assumptions in complexity theory (as the conjecture implies $\mathsf{P} \neq \mathsf{PSPACE}$). In this sense, our result only proves Grohe's conjecture modulo complexity theoretical assumptions and the additional conditions on strongly unboundedness necessitated by this. On the other hand, our result is stronger than the conjecture in that we only require a fixed log-power rather than polylog.

In [MM03], Makowsky and Mariño study similar questions in relation to classes of graphs closed under topological minors. They show that any such class must have bounded treewidth for MSO_2 model-checking to be in FPT. Closure under topological minors is a much stronger condition simplifying the proof significantly. However, in the same paper, the authors give examples for classes of graphs of unbounded cliquewidth but with tractable MSO_1 model-checking. These examples can be adapted to examples of classes of graphs which are closed under subgraphs, whose treewidth is only bounded logarithmically (but which almost have logarithmic treewidth) and on which MSO_2 model-checking is tractable. This shows that in full generality, our results can not be strengthened much beyond the $\log^{28\gamma} n$ bound postulated in Theorem 8.3.

On Strongly Unbounded Treewidth

Let us give some justification for the two conditions in Definition 8.1. The first condition is a consequence of the fact that we prove our main result by reducing an NP-hard

[2] The original conjecture is formulated in terms of branchwidth but this is equivalent.

problem to $\mathrm{MC}(\mathrm{MSO}_2, \mathcal{C})$. Without this condition there could simply be too few graphs of high treewidth in $\mathcal{C}$ to define a reduction. To give an example, fix a constant c and let H_n be the graph constructed from the $n \times n$-grid by replacing every edge by a path on $\frac{1}{m} \cdot 2^{\sqrt[c]{n}}$ vertices, where $m = n^2$. The resulting graph has $\mathcal{O}(2^{\sqrt[c]{n}})$ vertices and treewidth n. Now let $\mathcal{C}' := \{H_n : n = 2^{2^i}, i > 0\}$ and let $\mathcal{C}$ be the subgraph closure of $\mathcal{C}'$. If $c > 29$, then the treewidth of $\mathcal{C}$ is unbounded by $\log^{29} n$ but not strongly unbounded by this function, while being closed under taking subgraphs. To see this, take a graph $H_n \in \mathcal{C}'$, for some $n = 2^{2^i}$, $i > 2$. Any subgraph $H \subseteq H_n$ is either acyclic, and therefore has treewidth 1, or it contains a path of length $\frac{1}{m} \cdot 2^{\sqrt[c]{n}}$. Thus, H_n does not contain any subgraph $H \subseteq H_n$ of treewidth $2^i \leq \mathrm{tw}(H) \leq p(2^i)$ such that $\mathrm{tw}(H) > \log^c |H|$, for any fixed polynomial p. It follows that if we wanted to use $\mathcal{C}$ for a reduction as outlined below, there wouldn't be enough graphs of large treewidth to reduce to: given an instance of Sat of length 2^i for an i that is not close to a power of 2, we would have no chance in identifying a graph in $\mathcal{C}$ to perform a reduction in polynomial time. Therefore, as long as we have to rely on reductions to prove results as in this work, a condition similar to Condition 1 seems necessary. The second condition is necessary to prevent artificial cases where constructing a graph in the class $\mathcal{C}$ is already so expensive that any reduction would take too much time.

Overview of the Proof

Let us briefly sketch the main ideas of the proof, the basic framework of which is adapted from [Kre09b]. Let $\mathcal{C}$ be a class of graphs with treewidth strongly unbounded by $\log^c n$, for some suitable c.

We aim at reducing the propositional satisfiability problem Sat to $\mathrm{MC}(\mathrm{MSO}_2, \mathcal{C})$. Towards this aim we will first construct an MSO_2-formula φ, depending only on a Turing machine deciding Sat, and then, given a Sat-instance w, construct a graph $G_w \in \mathcal{C}$ such that $G_w \models \varphi$ if and only if w is satisfiable. The idea is to encode the instance w in the graph G_w so that (i) the instance can be decoded by the MSO_2-formula φ and (ii) the graph G_w contains enough structure so that the formula φ can simulate the run of a Turing machine deciding SAT on input w.

Similar ideas in connection with treewidth have been employed in the past and the usual approach is to use the the Excluded Grid Theorem 7.3 of Robertson, Seymour, and Thomas [RST94] that there is a function $f : \mathbb{N} \to \mathbb{N}$ such that every graph of treewidth $f(k)$ contains a $k \times k$-grid as a minor. Such a grid provides enough structure to simulate runs of Turing machines in MSO_2 and encoding the Sat instance w in a grid can easily be done by deleting certain edges (see Section 8.2).

However, the best known bound for the function f known to date is exponential and as we are dealing with graphs of treewidth only logarithmic in the number of vertices, the grids we are guaranteed to find in this way are essentially only of order $\log \log |G_w|$ which is much too small for any reduction to work.

Instead of using grids, therefore, we will use a new structural characterization of treewidth developed by Reed and Wood [RW08] and made algorithmic in Chapter 7 of this work which replaces grids by *grid-like minors*. It was shown in [RW08] that any

graph contains a grid-like minor of order polynomial in its treewidth and we showed in Theorem 7.27 that these are computable in polynomial time. The main problem with grid-like minors is that they do not occur as minors of the graph itself but only of the intersection graph of sets of pairwise disjoint paths (see Section 7.2). As indicated above, we would like to encode a SAT-instance w in a grid by deleting certain edges. But as grid-like minors only occur as minors of intersection graphs, deleting an edge in a graph G has no predictable implication for the grid-like minor which makes encoding SAT-instances using grid-like minors extremely difficult.

Therefore, instead of encoding SAT-instances in grid-like minors directly, we will encode them in a *labeled tree-ordered web* as defined in Section 7.4 and illustrated in Figure 7.5. Such a structure essentially consists of a grid-like minor attached to a special tree T such that there is an MSO_2-formula defining a linear order on the vertices of T (see Section 8.1). Furthermore, the particular structure of the tree allows us to encode the letters w_i of a SAT-instance $w := w_1 \dots w_l$ as *crosses* and *double crosses* attached to some vertices of T. Hence, the order imposed on T together with the ability to encode letters allows us to encode the SAT-instance w in T. This labeling can, in turn, be transferred to a unique labeling of the grid-like minor. Hence, we will use this external tree to encode the SAT-instance and the grid-like minor as the structure we need to simulate the run of a Turing machine on the encoded input.

Finally, as we assume that the class $\mathcal{C}$ of graphs we work in is closed under subgraphs, this labeled tree-ordered web occurs as a graph in $\mathcal{C}$. Hence, if evaluating the MSO_2-formula which decodes the encoded SAT-instance and simulates the run of a Turing machine on it was fixed-parameter tractable, we could solve SAT in subexponential time.

Outline of this Chapter

We start by showing how to define the various parts and the order of a labeled tree-ordered web in MSO_2. Afterwards, we review the notion of MSO_2–MSO_2-interpretations and the hardness of MSO_2 on colored walls before showing its hardness on uncolored walls. Finally, we show how to define an interpretation of a labeled tree-ordered web in a colored wall and present the proof of Theorem 8.3.

8.1 Defining a Labeled Tree-Ordered Web in MSO_2

In this section, we aim at defining the various parts of a labeled tree-ordered web in MSO_2. We start by stating some auxiliary formulas that we need in our construction of the main formulas. We make use of the notation and basic formulas introduced in Section A.4 using the incidence structure encoding of graphs.

- $\mathit{adj}(v, w, H) := v \neq w \wedge \exists e \in H(v \in e \wedge w \in e)$ says v and w are adjacent in H;
- $\mathit{pathends}(v, w, P) := \mathit{path}(P) \wedge \mathit{deg}^{=1}(v, P) \wedge \mathit{deg}^{=1}(w, P)$ says P is a path with endpoints v and w;
- $\mathit{topneigh}(v, w, P, H) := P \subseteq H \wedge \mathit{pathends}(v, w, P) \wedge \forall z \in V(P)(z \neq v \wedge z \neq w \rightarrow \mathit{deg}^{=2}(z, H))$ says u and w are topological neighbors connected by P in H;
- $\mathit{leaf}(v, T) := \mathit{deg}^{=1}(v, T)$ says v is a leaf of T;
- $\mathit{leafy}(v, T) := \mathit{deg}^{=3}(v, T) \wedge \exists w(\mathit{adj}(v, w, T) \wedge \mathit{leaf}(w, T))$ says v is leafy in T;
- $\mathit{pbranch}(v, T) := \mathit{deg}^{=3}(v, T) \wedge \neg \mathit{leafy}(v, T)$ says v is a proper branching vertex;

Henceforth, we assume T is a tree and that a formula $\mathit{root}(r, T)$ is given.

- $\mathit{ancstr}(v, a, T) := \exists r\big(\mathit{root}(r, T) \wedge (r = a \vee \exists P \subseteq T\,(\mathit{pathends}(v, r, P) \wedge a \in V(P)))\big)$ says a is an ancestor of v in T;
- $\mathit{cca}(v, w, a, T) := \mathit{ancstr}(v, a, T) \wedge \mathit{ancstr}(w, a, T) \wedge \neg\exists a'\big(a \neq a' \wedge \mathit{ancstr}(v, a', T) \wedge \mathit{ancstr}(w, a', T) \wedge \mathit{ancstr}(a', a, T)\big)$ says a is the closest common ancestor of v and w in T;
- $\mathit{parent}(v, p, T) := \mathit{ancstr}(v, p, T) \wedge \mathit{adj}(v, p, T)$ says p is the parent of v in T;
- $\mathit{child}(v, c, T) := \mathit{adj}(v, c, T) \wedge \neg\mathit{parent}(v, c, T)$ says c is a child of v in T;
- $\mathit{subtree}(v, H, T) := H \subseteq T \wedge v \in V(H) \wedge \forall p\big(\mathit{parent}(v, p, T) \rightarrow p \notin V(H) \wedge \forall e \in T(p \notin e \wedge e \cap V(H) \neq \varnothing \rightarrow e \in H\big)$ expresses for any vertex v other than the root of T that H is the subtree of T rooted at v.
- $\mathit{extsubtree}(v, H, T) := \exists c, e, H'\, \mathit{child}(v, c, T) \wedge \mathit{subtree}(c, H', T) \wedge c \in e \wedge v \in e \wedge H = H' \cup e$ says H is the extended subtree of a child of v in T;

Lemma 8.5. *There exists a uniform* MSO_2*-formula* $\varphi_{\preceq}(x, y, T)$ *which defines the canonical order* $\leq_T$ *on any rooted subcubic tree* (T, r) *and set* $X \subseteq V(T)$ *in a graph* G*, assuming that* (T, r, X) *admits a definable order and that* MSO_2*-formulas* $\varphi_R(v, T)$ *and* $\varphi_X(v)$ *defining the root of* T *and the set* X*, respectively, are given.*

Proof. We have to define the conditions of Definition 7.49 in MSO_2. The first two conditions are easily captured by the following formula:

$$\begin{aligned}\mathit{left}_1(T_1, T_2) := {} & \big(\exists^{=1} e(e \in T_1) \wedge \exists^{\geq 2} e(e \in T_2)\big) \vee \big(\mathit{path}(T_1) \wedge \neg\mathit{path}(T_2)\big) \vee \\ & \big(\exists^{=1} e\big(e \in T_1 \wedge \exists v(v \in e \wedge \varphi_X(v))\big) \wedge \exists^{=1} e(e \in T_2)\big)\end{aligned}$$

For two subgraphs T_1 and T_2, this formula says that if T_1 is a single edge but T_2 is not, or if T_1 is a simple path and T_2 is not, or T_1 and T_2 are both single edges and T_1 contains a good vertex, then T_1 is left of T_2. To capture the third condition, we need to compare the parity of good or leafy vertices on a path. So, let $\mathit{gl}(v) := \varphi_X(v) \vee \mathit{leafy}(v)$. First,

we define an auxiliary formula expressing that a vertex v has a topological neighbor w such that none of the internal vertices of the subpath from v to w are good or leafy:

$$\begin{aligned}&\mathit{topneigh}_{gl}(v, w, T) :=\\ &\qquad \exists P \subseteq T\big(\mathit{pathends}(v, w, P) \wedge \neg\exists z \in V(P)\,(z \neq v \wedge z \neq w \wedge \mathit{gl}(z))\big)\end{aligned}$$

Consider the following formula that guarantees that a path P from v to w contains an odd number of good or leafy vertices, where we assume v is neither good nor leafy:

$$\begin{aligned}&\mathit{odd}_{gl}(P, v, w) := \exists C_1, C_2 \subseteq V(P)\\ &\quad \big(C_1 \cap C_2 = \varnothing \wedge \forall x \in C_1 \cup C_2\ \mathit{gl}(x) \wedge \forall x \in V(P)(x \neq v \wedge \mathit{gl}(x) \rightarrow x \in C_1 \cup C_2) \wedge\\ &\quad \forall x \in C_2 \exists y_1, y_2 \in C_1 \big(y_1 \neq y_2 \wedge \mathit{topneigh}_{gl}(x, y_1, P) \wedge \mathit{topneigh}_{gl}(x, y_2, P)\big) \wedge\\ &\quad \forall x \in C_1 \exists y_1, y_2 \big((y_1 = v \vee y_1 \in C_2) \wedge (y_2 = w \vee y_2 \in C_2) \wedge y_1 \neq y_2 \wedge\\ &\qquad\qquad \mathit{topneigh}_{gl}(x, y_1, P) \wedge (\mathit{topneigh}_{gl}(x, y_2, P) \vee x = w)\big) \wedge\\ &\quad v \notin C_1 \cup C_2 \wedge \exists x \in C_1\ \mathit{topneigh}_{gl}(v, x)\big)\end{aligned}$$

The idea is to color the good or leafy vertices on P with two colors C_1 and C_2, such that a vertex of one color has only vertices of the other color as a direct topological neighbor. Now if we guarantee that the first and last vertex are colored C_1, we have an odd number of good or leafy vertices on P. The next formula says that w is the closest proper branching vertex to v in a given subtree H connected to v by the path P:

$$\begin{aligned}\mathit{closest\text{-}pbv}(v, w, P, H) := \mathit{pathends}(v, w, P) \wedge \mathit{pbranch}(w, H) \wedge\\ \neg\exists z \in V(P)\,\big(z \neq v \wedge z \neq w \wedge \mathit{pbranch}(z, H)\big)\end{aligned}$$

Similarly, we can define the property that w is the farthest good leaf from v if no proper branching vertex occurs in H:

$$\begin{aligned}&\mathit{farthest\text{-}leaf}(v, w, P, H) := \mathit{pathends}(v, w, P) \wedge \mathit{leaf}(w, H) \wedge \varphi_X(w) \wedge\\ &\qquad \forall z \in V(P)\,\big(z \neq v \wedge \mathit{deg}^{=3}(z, H) \rightarrow \exists y(y \neq w \wedge \mathit{leaf}(y, H) \wedge \mathit{adj}(z, y, H))\big)\end{aligned}$$

Now if T_1 and T_2 are the extended subtrees of the children of a branching vertex v, we can determine if T_1 is left of T_2 according to Definition 7.49 as follows:

$$\begin{aligned}&\mathit{left}(v, T_1, T_2) := \mathit{left}_1(T_1, T_2) \vee \big(\neg \mathit{left}_1(T_2, T_1) \wedge\\ &\quad \exists w, P \subseteq T_1\,\big((\mathit{closest\text{-}pbv}(v, w, P, T_1) \vee \mathit{farthest\text{-}leaf}(v, w, P, T_1)) \wedge \mathit{odd}_{gl}(P, v, w)\big)\big)\end{aligned}$$

Finally, we can define the canonical order $\leq_T$:

$$\begin{aligned}\varphi_{\leq}(x, y, T) := \exists v\big(\mathit{cca}(x, y, v, T) \wedge \big(v = x \vee (v \neq y \wedge\\ \exists T_1, T_2 \subseteq T\,(\mathit{extsubtree}(v, T_1, T) \wedge \mathit{extsubtree}(v, T_2, T) \wedge\\ x \in V(T_1) \wedge y \in V(T_2) \wedge \mathit{left}(T_1, T_2)))\big)\big)\end{aligned}$$

□

Lemma 8.6. *Given a labeled tree-ordered web $\mathcal{W} = (G, T, r, A, \mathcal{P}, \mathcal{Q}, X, C)$, there exist* MSO_2*-formulas $\varphi_T(H)$, $\varphi_R(v)$, $\varphi_A(v)$, $\varphi_X(v)$, $\varphi_{Cr_1}(H)$, $\varphi_{Cr_2}(H)$,$\varphi_{PQ}(H)$, and $\varphi_{\preceq}(x, y)$ defining the tree T, its root r, the set of good vertices A, the bases of the crosses X, the single and double crosses C, the edge set of the grid-like minor $\bigcup\mathcal{P} \cup \bigcup\mathcal{Q}$, and the canonical order $\leq_T$, respectively. These formulas are uniform and do not depend on $\mathcal{W}$ in any form.*

Proof. We construct the required formulas gradually, occasionally using auxiliary formulas; for formulas that we defined previously as $\varphi(\cdot, H)$, we sometimes write simply $\varphi(\cdot)$ for $\varphi(\cdot, E)$:

- $\mathit{rootish}(v) := \mathit{deg}^{=3}(v) \wedge \exists x, y\,(\mathit{leaf}(x) \wedge \mathit{leaf}(y) \wedge \mathit{adj}(v, x) \wedge \mathit{adj}(v, y))$ says that v is of degree 3 and has two neighbors of degree 1;
- $\varphi_R(v) := \mathit{rootish}(v) \wedge \exists w, P\,(\mathit{topneigh}(v, w, P) \wedge \mathit{pbranch}(w))$ uniquely defines the root of the tree T;
- $\mathit{leafytopneigh}(v, w, P) := \mathit{topneigh}(v, w, P) \wedge \mathit{leafy}(w)$ says that w is leafy topological neighbor of v connected by path P;
- $\mathit{cross}_1(x, y, b, P_1, P_2) := \mathit{rootish}(x) \wedge \mathit{leafytopneigh}(x, y, P_1) \wedge \mathit{topneigh}(y, b, P_2) \wedge \mathit{pbranch}(b)$ says that b is the base of a single cross, x its tail, y its leafy vertex in the middle, and P_1 and P_2 the paths connecting x, y, and b, respectively;
- $\varphi_{Cr_1}(H) := \exists x, y, b \subseteq V(H) \exists P_1, P_2 \subseteq H\,(\mathit{cross}_1(x, y, b, P_1, P_2) \wedge \forall e \in H$ $(e \in P_1 \vee e \in P_2 \vee x \in e \vee y \in e) \wedge \forall e(x \in e \vee y \in e \to e \in H))$ defines a single cross of $\mathcal{W}$; the formula $\varphi_{Cr_2}(H)$ defining a double cross can be obtained analogously; note that since the leaves of T are marked with leaf-marks and X lies flat in T, the bases of the crosses are not leafy – this is crucial for making the crosses definable;
- the formula $\varphi_X(v)$ defining the bases of the crosses is immediately derived using cross_1 and its analog cross_2;
- $\mathit{spec}(v) := \mathit{deg}^{=4}(v) \vee \mathit{deg}^{=3}(v) \wedge \neg\exists w, P\,(\mathit{topneigh}(v, w, P) \wedge \mathit{leafy}(w))$ defines the set of *special* vertices of G, i.e. the vertices of degree 4 or 3 that do not have a leafy vertex as a topological neighbor;
- $\varphi_A(v) := \mathit{deg}^{=3}(v) \wedge \neg\mathit{spec}(v) \wedge \exists w, P\,(\mathit{topneigh}(v, w, P) \wedge \mathit{spec}(w))$ defines the good vertices of the tree as given in Definition 7.52;
- $\varphi_{PQ}(H) := \forall v, e(\varphi_A(v) \wedge v \in e \to (\exists u, P(\mathit{topneigh}(v, u, P) \wedge \mathit{spec}(u) \wedge e \in P) \leftrightarrow e \in H)) \wedge \forall v, e(\neg\varphi_A(v) \wedge v \in e \wedge v \in V(H) \to e \in H) \wedge \mathit{conn}(H)$ defines the edges of the grid-like minor $(\mathcal{P}, \mathcal{Q})$ by specifying that of the edges adjacent to a good vertex exactly the one that starts a good path belongs to $(\mathcal{P}, \mathcal{Q})$; the definition is completed by taking a maximal connected subgraph that includes these edges;
- $\mathit{leafmark}(H) := \exists e_1, e_2, x, y, z\,(H = \{e_1, e_2\} \wedge e_1 = \{x, y\} \wedge e_2 = \{y, z\} \wedge \mathit{leaf}(x) \wedge \mathit{deg}^{=2}(y))$ defines a leaf-mark according to Definition 7.52;

- $\mathit{rootmark}(H) := \exists e_1, e_2, r, x, y\,(H = \{e_1, e_2\} \wedge e_1 = \{r, x\} \wedge e_2 = \{r, y\} \wedge \mathit{leaf}(x) \wedge \mathit{leaf}(y))$ defines the marking of the root;
- $\varphi_T(H) := \forall e(e \in H \leftrightarrow \neg\exists H'(e \in H' \wedge (\mathit{rootmark}(H') \vee \mathit{leafmark}(H') \vee \varphi_{Cr_1}(H') \vee \varphi_{Cr_2}(H') \vee \varphi_{PQ}(H'))))$ uniquely defines the tree T as the set of edges that do not belong to markings, crosses, or the grid-like minor.

Finally, we obtain $\varphi_{\leq}(x, y) := \exists T\,(\varphi_T(T) \wedge x \in V(T) \wedge y \in V(T) \wedge \varphi_{\leq}(x, y, T))$ where $\varphi_{\leq}(x, y, T)$ denotes the formula obtained from Lemma 8.5. □

8.2 MSO_2 Interpretations and Walls

In this section, we first show the intractability of MSO_2 on walls and then lift this to show the general result. For this, we first recall the well-known fact that MSO_2 is intractable on colored walls. Recall that from the results of the previous chapter, given a word w and a graph G of large enough treewidth, we construct either a wall encoding w or a labeled tree-ordered web encoding w. For either outcome we will define an MSO_2-interpretation of colored walls in these structures which will allow us to transfer the intractability results from colored walls to these structures.

8.2.1 MSO_2-Interpretations

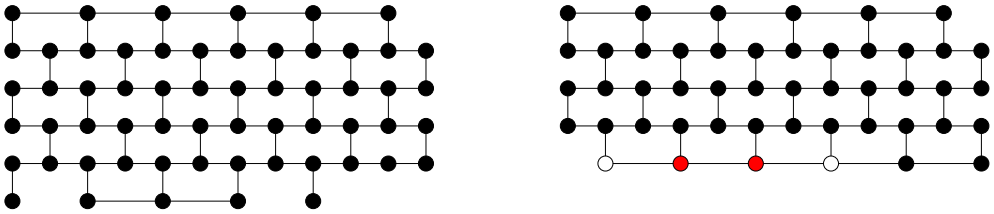
We first recall briefly the concepts of interpretations (see e.g. [Hod97]). In logic, they play a similar role to many-one reductions in complexity theory.

Definition 8.7. Let σ and τ be signatures and let $\overline{X}$ be a tuple of monadic second-order variables. An *interpretation of τ in σ with parameters $\overline{X}$* is a tuple $\Theta := (\varphi_{\mathit{valid}}, \varphi_{\mathit{univ}}(x), \varphi_{\sim}(x, y), (\varphi_R(\overline{x}))_{R \in \tau})$ of $\mathrm{MSO}_2[\sigma \dot{\cup} \overline{X}]$-formulas, where the arity of $\overline{x}$ in $\varphi_R(\overline{x})$ is $\mathit{ar}(R)$, such that for all σ-structures A and assignments $\overline{Y} \subseteq U(A)$ to $\overline{X}$ with $(A, \overline{Y}) \models \varphi_{\mathit{valid}}$, $\varphi_{\sim}$ defines an equivalence relation on $\varphi_{\mathit{univ}}(A)$.

For an interpretation Θ we will denote φ_{valid} by $\varphi_{\mathit{valid}}(\Theta)$. With any interpretation Θ we associate a map taking a σ-structure A and $\overline{Y} \subseteq U(A)$ such that $(A, \overline{Y}) \models \varphi_{\mathit{valid}}$ to a τ-structure H with universe $U(H) := \varphi_{\mathit{univ}}(A, \overline{Y})_{|\varphi_{\sim}(A,\overline{Y})} := \{[v]_{\sim} : (A, \overline{Y}) \models \varphi_{\mathit{univ}}(v)\}$ where $[v]_{\sim}$ denotes the equivalence class of v under $\varphi_{\sim}(A, \overline{Y})$. For $R \in \tau$ of arity $r := \mathit{ar}(R)$ we define $R(H) := \{([a_1], \ldots, [a_r]) : (A, \overline{Y}) \models \varphi_R(a_1, \ldots, a_r)\}$. For given $(A, \overline{Y})$, we denote the resulting τ-structure by $\Theta(A, \overline{Y})$.

Furthermore, any interpretation Θ also defines a translation of $\mathrm{MSO}_2[\tau]$-formulas φ to $\mathrm{MSO}_2[\sigma]$-formulas $\Theta(\varphi)$ by replacing occurrences of relations $R \in \tau$ by their defining formulas $\varphi_R \in \Theta$ in the usual way (see [Hod97] for details) so that the following lemma holds. From now on we will always let σ and τ be σ_{graph} or expansions thereof and therefore speak about interpretations without any reference to specific signatures.

Lemma 8.8 (Interpretation Lemma)**.** *Let Θ be an MSO_2-interpretation with parameters $\overline{X}$. For any σ_{graph}-structure A and assignment $\overline{Y} \subseteq U(A)$ to $\overline{X}$ s.t. $(A, \overline{Y}) \models \varphi_{\mathit{valid}}(\Theta)$, and any MSO_2-sentence φ we have $\Theta(A, \overline{Y}) \models \varphi$ if, and only if, $(A, \overline{Y}) \models \Theta(\varphi)$.*



(a) (b)

Figure 8.1: (a) A wall-encoding of 0110 and (b) the colored wall it is interpreted as.

8.2.2 MSO_2 on Colored Elementary Walls

The signature σ_{wall} of colored walls is defined as $\sigma_{wall} := \{V, E, \in, C_0, C_1\}$, where V, E, C_0, C_1 are unary relation symbols and $\in$ is a binary relation symbol. A σ_{wall}-structure W is a *colored elementary $\ell \times \ell$-wall* if its σ_{graph}-reduct $W_{|\{V,E,\in\}}$ is an elementary $\ell \times \ell$-wall according to Definition 7.38. W encodes a word $w := w_1 \dots w_n \in \Sigma^n$ with power d if $\ell > n^d$ and if $\{v_{1,i} : 1 \leq i \leq \ell\}$ are the vertices on the bottom row then $v_{1,i} \in C_0$ if and only if $w_i = 0$ and $v_{1,i} \in C_1$ if and only if $w_i = 1$, for all $1 \leq i \leq n$, and $C_0 \cup C_1 = \{v_{1,i} : 1 \leq i \leq n\}$ (see Figure 8.1 (b)).

The following lemma, whose proof is standard, is part of the folklore and immediately implies Theorem 8.10 below.

Lemma 8.9. *Let M be a nondeterministic n^d-time bounded Turing machine. There is a formula $\varphi_M \in \mathrm{MSO}_2$ such that for all words $w \in \Sigma^\star$, if W is a colored elementary wall encoding w with power d, then $W \models \varphi_M$ if, and only if, M accepts w. Furthermore, the formula φ_M can be constructed effectively from M. The same holds if M is an alternating Turing machine with a bounded number of alternations, as they are used to define the polynomial-time hierarchy.*

Theorem 8.10. *For $d \geq 2$ let $\mathfrak{W}_d$ be the class of colored elementary walls encoding words with power d. Then $\mathrm{MC}(\mathrm{MSO}_2, \mathfrak{W}_d)$ is not in XP unless $\mathsf{P} = \mathsf{NP}$.*

8.2.3 MSO_2 on Uncolored Walls

The previous paragraph stated the intractability of MSO_2 on colored elementary walls. As one possible outcome of Theorem 7.58 we get an uncolored wall W, not necessarily elementary, of sufficient size. In the absence of colors we will encode a word w in W by taking a suitable subgraph $W_{enc} \subseteq W$ as follows.

Let $w := w_1, \dots, w_n \in \{0,1\}^*$ be a word of length n, let $d \geq 1$ and let $m := n^d + 1$. The aim is to encode w in a wall W of order at least $m \times m$. Let $v_1, \dots, v_{m+1}$ be the nails (see 7.38) on the bottom row $B \subseteq W$ of W and, for $1 \leq i \leq m$, let $P_i \subseteq B$ be the subpath connecting v_i and v_{i+1}. Let $W_{enc} \subseteq W$ be the subgraph obtained from W by deleting the vertices $v_{n+2}, \dots, v_{m+1}$ and the internal vertices and edges of P_i for each $1 \leq i \leq n$ with $w_i = 0$. All other paths remain unchanged. We say that W' is a *wall-encoding of w with power d*. Figure 8.1 (a) shows a wall-encoding of the word `0100`

with power 1. Note that by deleting the vertices $v_{n+2}, \ldots, v_{m+1}$ we ensure that the left side of W_{enc} is uniquely identified.

It is well-known, see e.g. [Kre09b], that a wall W can be defined by an MSO_2-formula $\varphi_{wall}(W, \mathcal{R}, \mathcal{C}, B, L, T, R, N)$ expressing that

(i) $\mathcal{R}$ and $\mathcal{C}$ are sets of edges consisting of disjoint paths such that $W = \mathcal{R} \cup \mathcal{C}$; we think of $\mathcal{R}$ as the set of rows and of $\mathcal{C}$ as the set of columns of W; note that an $m \times n$-wall has exactly $m + 1$ rows and $n + 1$ columns and each column and row have exactly two nails in common, except on the top and bottom rows, where they intersect in only one nail;

(ii) $B, T \subseteq \mathcal{R}$ are the bottom and top row of W and $L, R \subseteq \mathcal{C}$ are the leftmost and rightmost column of W, respectively; and

(iii) $N \subseteq V(W)$ is the set of nails of W.

Sometimes we use $\varphi_{wall}(W) := \exists \mathcal{R}, \mathcal{C}, B, L, T, R, N\, \varphi_{wall}(W, \mathcal{R}, \mathcal{C}, B, L, T, R, N)$ as a shortcut. Furthermore, we need the formula $\mathit{dpaths}(H) := \mathit{ac}(H) \wedge \forall v \in V(H)\ \mathit{deg}^{\leq 2}(v)$, which expresses that H is a set of edges consisting of a number of disjoint paths, and the formula $\mathit{pathof}(P, u, v, H) := P \subseteq H \wedge \mathit{pathends}(u, v, P) \wedge \mathit{leaf}(u, H) \wedge \mathit{leaf}(v, H)$ expressing that P is a path with endpoints v and w of degree 1 in H; in particular, if H is a set of disjoint paths, this formula asserts that P is one of the disjoint paths in H; we use the shortcut $\mathit{pathof}(P, H) := \exists u, v\ \mathit{pathof}(P, u, v, H)$. We can now prove

Theorem 8.11. *There is an* MSO_2*-interpretation* Θ *of* σ_{wall} *in* σ_{graph} *such that if* W_{enc} *is an uncolored wall-encoding of order at least* 4 *of* $w \in \Sigma^*$ *with power* d *then* $\Theta(W_{enc})$ *is a colored elementary wall encoding* w *with power* d*.*

Proof. We can think of an $m \times m$ wall-encoding W_{enc} of $w = w_1 \ldots w_n$ as an $(m-1) \times m$ wall W augmented by a set of edges M that we call the *marking of the wall*. We require that the marking is attached only to the bottom row and only to nonnail vertices or to the first, i.e. leftmost vertex on the bottom row. We define the interpretation Θ with parameters W, B, L, R, N, and M as follows. To this end, we make use of a formula φ_{bwall} expressing that W is a wall in which the left and right columns each contain 3 or more nails of global degree 2:

$$\varphi_{bwall}(W, B, L, R, N) := \exists \mathcal{R}, \mathcal{C}, T\, \varphi_{wall}(W, \mathcal{R}, \mathcal{C}, B, L, T, R, N) \wedge$$
$$\exists^{\geq 3} v \in N \cap V(L)\ \mathit{deg}^{=2}(v) \wedge \exists^{\geq 3} v \in N \cap V(R)\ \mathit{deg}^{=2}(v)$$

If the given graph is a wall-encoding, this makes sure that the wall spans from the left side to the right side of the wall-encoding and furthermore, cannot contain any edge of the marking of the wall, since any wall that contains such edges cannot contain 3 or more nails of degree 2 on its right boundary. Again, we define the shortcut $\varphi_{bwall}(W)$

similar to the case of $\varphi_{wall}(W)$. Now we can define the formula $\varphi_{valid}(\Theta)$:

$$\begin{aligned}\varphi_{valid}(W,B,L,R,N,M) := {} & \varphi_{bwall}(W,B,L,R,N) \wedge \\ & \forall W'(\varphi_{bwall}(W') \rightarrow W' \subseteq W) \wedge \\ & \forall e(e \in M \leftrightarrow e \notin W) \wedge \\ & \forall v \in V(M) \cap V(W)\,(v \in V(B) \wedge (v \in V(L) \vee v \notin N))\end{aligned}$$

Let $W_{col} := \Theta(W_{enc})$ be the colored wall we aim at in the interpretation. We define the set of vertices of W_{col} simply as the set of nails of W: $\varphi_V(x) := x \in N$.

The edges of W_{col} are the equivalence classes of subdivided edges of W:

$$\begin{aligned}\varphi_E(x) &:= e \in W \\ \varphi^e_{eq}(x,y) &:= \exists u, v \in N\, \exists P \subseteq W\,(x \in P \wedge y \in P \wedge \mathit{pathends}(u,v,P) \wedge \\ &\qquad \forall z \in V(P)\,(z = u \vee z = v \vee z \notin N))\end{aligned}$$

Now, we obtain the universe, equivalence relation, and incidence relation of Θ as:

$$\begin{aligned}\varphi_{univ}(x) &:= \varphi_V(x) \vee \varphi_E(x) \\ \varphi_{eq}(x,y) &:= \varphi^e_{eq}(x,y) \\ \varphi_{\in}(v,e) &:= \varphi_V(v) \wedge \varphi_E(e) \wedge \exists e' \in W\,(v \in e' \wedge \varphi^e_{eq}(e,e'))\end{aligned}$$

It remains to define the colors of W_{col}. For a nail v on the bottom row of the wall, consider the closest vertices u and w to its left and right that are incident to an edge of the marking, if existent. Now v is to be interpreted as a 0 if and only if u and w belong to *different* connected components of the marking; and v is to be interpreted as a 1 if u and w belong to the *same* connected component of the marking. In formulas, we have

$$\begin{aligned}\mathit{markingof}(x,X,Y) := {} & x \in N \cap V(B)\, \exists u, v\, \exists P \subseteq B \\ & (\mathit{pathends}(u,v,P) \wedge u \neq v \wedge x \in V(P) \\ & \mathit{components}(X,M) \wedge \mathit{components}(Y,M) \wedge u \in V(X) \wedge v \in V(Y) \wedge \\ & \forall z \in V(P)\,(z = v \vee z = u \vee \neg \exists e \in M\ z \in e)) \\ \varphi_{C_0}(x) := {} & \exists X, Y\,(\mathit{markingof}(x,X,Y) \wedge X \neq Y) \\ \varphi_{C_1}(x) := {} & \exists X, Y\,(\mathit{markingof}(x,X,Y) \wedge X = Y)\end{aligned}$$

This finishes the definition of the interpretation Θ. □

8.2.4 MSO_2 on Labeled Tree-Ordered Webs

The aim of this section is to show that we can define a colored elementary wall encoding a word w in a labeled tree-ordered web encoding w. The proof follows the basic ideas of a related proof by Kreutzer [Kre09b].

Theorem 8.12. *There is an* MSO_2*-interpretation* Θ *such that if* $(G, T, r, A, \mathcal{P}, \mathcal{Q}, X, C)$ *is a labeled tree-ordered web encoding a word* w *with power* d*, then* $\Theta(G)$ *is a colored elementary wall encoding* w *with power* d*.*

Proof. We will define the interpretation in a sequence of steps and will illustrate the formulas by a labeled tree-ordered web $\mathcal{W} = (G, T, r, A, \mathcal{P}, \mathcal{Q}, X, C)$ encoding a word w of length n with power d. The actual formulas will not depend on $\mathcal{W}$ in any form.

By Lemma 8.6, there exist MSO_2-formulas $\varphi_T(X)$, $\varphi_R(x)$, $\varphi_A(x)$, $\varphi_X(x)$, $\varphi_{Cr_1}(X)$, $\varphi_{Cr_2}(X)$, $\varphi_{PQ}(X)$, and $\varphi_{\preceq}(x, y)$ defining T, r, A, X, the single and double crosses of C, the edges of the grid-like minor $(\mathcal{P}, \mathcal{Q})$, and the canonical order $\leq_T$, respectively. Essentially, we now have formulas which, on G as above, define the labeled tree-ordered web $\mathcal{W}$. As a shortcut, we define $\varphi^e_T(x) := \exists T \varphi_T(T) \wedge x \in T$ and $\varphi^v_T(x) := \exists T \varphi_T(T) \wedge v \in V(T)$ and do similarly for φ_{Cr_1}, φ_{Cr_2}, and φ_{PQ}.

What is left to do is to define formulas which generate a wall from the grid-like minor $(\mathcal{P}, \mathcal{Q})$ so that the bottom row of the wall is connected to the vertices in $A = \{v_1, \ldots, v_n\}$ in the correct order, where n is the length of the word w. Our interpretation Θ is defined with main parameters $\mathcal{P}$, $\mathcal{Q}$, and H that are intended to be the disjoint paths of the grid-like minor $(\mathcal{P}, \mathcal{Q})$ and a wall in the intersection graph $\mathcal{I}(\mathcal{P}, \mathcal{Q})$, respectively; furthermore, we require parameters B, L, N defining the bottom row, leftmost row, and the nails of H, respectively. Note that here we regard $\mathcal{P}$ and $\mathcal{Q}$ as sets of edges comprising disjoint paths each.

We start by defining $\mathcal{I} := \mathcal{I}(\mathcal{P}, \mathcal{Q})$ as follows. The *vertices* of $\mathcal{I}$ are equivalence classes of *edges* of G in the grid-like minor that appear in exactly one of $\mathcal{P}$ or $\mathcal{Q}$ and are equivalent if they belong to the same path in $\mathcal{P}$ or $\mathcal{Q}$. The *edges* of $\mathcal{I}$ are equivalence classes of *vertices* of G where two vertices are equivalent if they belong to the intersection of the same pair $P \in \mathcal{P}$ and $Q \in \mathcal{Q}$. Formally, let $\mathit{pathofPQ}(P) := \mathit{pathof}(P, \mathcal{P}) \vee \mathit{pathof}(P, \mathcal{Q})$ and

$$
\begin{aligned}
\varphi^{\mathcal{I}}_V(x) &:= (x \in \mathcal{P} \wedge x \notin \mathcal{Q}) \vee (x \in \mathcal{Q} \wedge x \notin \mathcal{P}) \\
\varphi^{\mathcal{I}}_E(x) &:= x \in V(\mathcal{P}) \cap V(\mathcal{Q}) \\
\varphi^{\mathcal{I},V}_{eq}(x, y) &:= \exists P \left(\mathit{pathofPQ}(P) \wedge \{x, y\} \subseteq P\right) \\
\varphi^{\mathcal{I},E}_{eq}(x, y) &:= \exists P, Q \left(\mathit{pathofPQ}(P) \wedge \mathit{pathofPQ}(Q) \wedge \{x, y\} \subseteq V(P) \cap V(Q)\right) \\
\varphi^{\mathcal{I}}_{eq}(x, y) &:= \varphi^{\mathcal{I},V}_{eq}(x, y) \vee \varphi^{\mathcal{I},E}_{eq}(x, y) \\
\varphi^{\mathcal{I}}_{\in}(x, y) &:= \varphi^{\mathcal{I}}_V(x) \wedge \varphi^{\mathcal{I}}_E(y) \wedge \exists x', y' (\varphi_{eq}(x, x') \wedge \varphi_{eq}(y, y') \wedge y' \in x')
\end{aligned}
$$

We would like to express that H is a wall in $\mathcal{I}(\mathcal{P}, \mathcal{Q})$. As described in the previous subsection, we have a formula φ_{wall} that asserts a graph to be a wall; if in this formula, we replace every occurrence of $x \in V$ by $\varphi^{\mathcal{I}}_V(x)$, every occurrence of $x \in E$ by $\varphi^{\mathcal{I}}_E(x)$, and every occurrence of $v \in e$ by $\varphi^{\mathcal{I}}_{\in}(v, e)$, we can derive a formula $\varphi^{\mathcal{I}}_{wall}(H, B, L, N)$ that asserts that H is a wall in $\mathcal{I}$ with bottom row B, leftmost row B, and nails N; the formula can be slightly adapted in such a way that for each equivalence class that is to be included in one of these sets, all representantives are present. Note that B and L are sets of vertices of G, and N is a set of edges of G.

Similarly, let $\varphi^{\mathcal{I}}_{pathends}(x, y, P)$ be a formula derived from the formula *pathends* defined earlier expressing that P is a path in $\mathcal{I}$ with endpoints x and y; here, x and y are edges of G is P is a set of vertices of G. For notational convenience, let us also write $x \in^{\mathcal{I}} V(Y)$ for the formula $\exists y \in Y\, \varphi^{\mathcal{I}}_{\in}(x, y)$ if x is a vertex of $\mathcal{I}$ and Y is a set of edges of $\mathcal{I}$. Now, we can define that a nail in B is left of another nail in B as follows;

$$\varphi^{\mathcal{I}}_{left}(x, y) := \exists z\, \exists P \subseteq B(z \in B \cap L \wedge \varphi^{\mathcal{I}}_{pathends}(z, y, P) \wedge x \in^{\mathcal{I}} V(P))$$

Next, we would like to define that a nail of H is attached to a good vertex $v \in A$:

$$\varphi_{attached}(x, v) := x \in N \wedge \varphi_A(v) \wedge \exists u, P(pathof(P, v, u, \mathcal{P}) \wedge x \in P)$$

Now we are ready to define $\varphi_{valid}(\Theta)$:

$$\begin{aligned}\varphi_{valid}(\mathcal{P}, \mathcal{Q}, H, B, L, N) := {} & dpaths(\mathcal{P}) \wedge dpaths(\mathcal{Q}) \wedge \exists Z\, (\varphi_{PQ}(Z) \wedge \mathcal{P} \cup \mathcal{Q} \subseteq Z) \wedge \\ & \varphi^{\mathcal{I}}_{wall}(H, B, L, N) \wedge \forall v\, (\varphi_A(v) \to \exists x\, (x \in N \wedge x \in V(B) \wedge \varphi_{attached}(x, v)) \wedge \\ & \forall x, y \in N\, (x \in V(B) \wedge \exists v\, \varphi_{attached}(x, v) \wedge \varphi^{\mathcal{I}}_{left}(y, x) \to \exists w\, \varphi_{attached}(y, w)) \wedge \\ & \forall x, y \in V(B)\, \forall u, v\, (\varphi_{attached}(x, u) \wedge \varphi_{attached}(y, v) \to (\varphi^{\mathcal{I}}_{left}(x, y) \leftrightarrow \varphi_{\leq}(u, v)))\end{aligned}$$

The first line says that $\mathcal{P}$ and $\mathcal{Q}$ are each sets of edges consisting of disjoint paths and that they are included in the grid-like minor of G; the second line asserts that H is a wall in the intersection graph $\mathcal{I}(\mathcal{P}, \mathcal{Q})$ and that all good vertices of the labeled tree-ordered web are attached to some nails of the first row of this wall; the third line ensures that for any nail that is attached to the tree, all the nails to its left are also attached; since there are exactly n good vertices and different nails can only be attached to different good vertices, this implies that exactly the first n nails on the bottom row of H are attached to the tree; finally, the last line asserts that attachments respect the order of the tree.

Now we can easily define the following parts of the interpretation Θ:

$$\begin{aligned}\varphi_V(x) &:= x \in N \\ \varphi_E(x) &:= x \in H \\ \varphi_{univ}(x) &:= \varphi_V(x) \vee \varphi_E(x) \\ \varphi_{eq}(x, y) &:= \varphi^{\mathcal{I}}_{eq}(x, y) \vee \exists u, v \in N\, \exists P \subseteq H\, (\varphi^{\mathcal{I}}_{pathends}(u, v, P) \wedge \\ &\qquad \forall z\, (z \in V(P) \to z = u \vee z = v \vee z \notin N) \wedge x \in V(P) \wedge y \in V(P)) \\ \varphi_{\in}(x, y) &:= \varphi_V(x) \wedge \varphi_E(y) \wedge \exists x', y'\, (\varphi_{eq}(x, x') \wedge \varphi_{eq}(y, y') \wedge y' \in x')\end{aligned}$$

Finally, it remains to define the colors, which we do as follows:

$$\begin{aligned}\varphi_{crossbase}(x, b) := {} & x \in N \wedge x \in V(B) \wedge \varphi_X(b) \wedge \exists v\, (\varphi_{attached}(x, v) \wedge \\ & \varphi_{\leq}(b, v) \wedge \forall u\, (\varphi_A(u) \wedge \varphi_{\leq}(u, v) \to \varphi_{\leq}(u, b))) \\ \varphi_{C_0}(x) := {} & \exists b \exists C\, (\varphi_{crossbase}(x, b) \wedge \varphi_{Cr_1}(C) \wedge b \in V(C)) \\ \varphi_{C_1}(x) := {} & \exists b \exists C\, (\varphi_{crossbase}(x, b) \wedge \varphi_{Cr_2}(C) \wedge b \in V(C))\end{aligned}$$

The formula $\varphi_{crossbase}$ determines if the given element x is a nail on the bottom row of the interpreted wall and is attached to some good vertex v; in this case, it ensures that b is the base of the cross immediately to the left of v in the tree. Then the formulas φ_{C_0} and φ_{C_1} simply check if this base b is incident to a single or a double cross. □

8.3 Proof of the Main Theorem

In this section we complete the proof of Theorem 8.3. We first prove Part 1.

Suppose, $\mathrm{MC}(\mathrm{MSO}_2, \mathcal{C}) \in \mathsf{XP}$, i.e. there is a computable function $f : \mathbb{N} \to \mathbb{N}$ such that given $G \in \mathcal{C}$ and $\varphi \in \mathrm{MSO}_2$ we can decide $G \models \varphi$ in time $\mathcal{O}(|G|^{f(|\varphi|)})$. Let M be a nondeterministic Turing machine deciding SAT in quadratic time and let φ_M be the formula constructible from M as defined in Lemma 8.9.

Let Θ_1 be the interpretation from the Theorem 8.11 and let Θ_2 be the interpretation from Theorem 8.12. Define $\varphi_M^1 := \Theta_1(\varphi_M)$ and $\varphi_M^2 := \Theta_2(\varphi_M)$.

Let $w \in \{0,1\}^*$ be a word of which we want to decide whether $w \in$ SAT, let $\ell := |w|$ and $t := 2c\ell^{28}$, where c is the constant from Theorem 7.58. As the treewidth of $\mathcal{C}$ is strongly unbounded by $\log^{28\gamma} n$, there are $\varepsilon < 1$ and a polynomial $p(n)$ of degree less than γ such that $\mathcal{C}$ contains a graph G with $\mathrm{tw}(G) \geq \log^{28\gamma} |G|$ and $t \leq \mathrm{tw}(G) \leq p(t)$ and G can be computed in time $2^{|w|^\varepsilon}$; note that this implies that

$$|G| \leq 2^{p(2c\ell^{28})^{\frac{1}{28\gamma}}} \leq 2^{|w|^\delta}, \text{ for some } \delta < 1\,.$$

By Theorem 7.58, G either contains a) a wall W_w encoding w with power 2 as a subgraph or b) a labeled tree-ordered web $\mathcal{W} = (H, T, r, A, \mathcal{P}, \mathcal{Q}, X, C)$ encoding w with power 2. Note that we need an encoding with power 2 because M needs $|w|^2$ space cells and computation steps to decide $w \in$ SAT.

In case a), as $\mathcal{C}$ is closed under subgraphs, $W_w \in \mathcal{C}$ and we can therefore decide $W_w \models \varphi_M^1$ in time

$$|W_w|^{f(|\varphi_M^1|)} \leq |G|^{f(|\varphi_M^1|)} \leq (2^{|w|^\delta})^{f(|\varphi_M^1|)} = 2^{f(|\varphi_M^1|)|w|^\delta} = 2^{o(|w|)}\,.$$

By construction, $W_w \models \varphi_M$ if, and only if, M accepts w if, and only if, $w \in$ SAT.

In case b), $H \in \mathcal{C}$ as H is a subgraph of G. We can therefore decide $H \models \varphi_M^2$ in time

$$|H|^{f(|\varphi_M^2|)} \leq |G|^{f(|\varphi_M^2|)} \leq (2^{|w|^\delta})^{f(|\varphi_M^2|)} = 2^{f(|\varphi_M^2|)|w|^\delta} = 2^{o(|w|)}$$

By construction, $H \models \varphi_M$ if, and only if, M accepts w if, and only if, $w \in$ SAT.

Hence, in both cases we can decide $w \in$ SAT in time $2^{o(|w|)}$. This shows Part 1.

To show Part 2, we use the same proof idea. Let P be a language in the polynomial-time hierarchy and let M be an alternating Turing machine with bounded alternation deciding P in time n^k. We use essentially the same proof as above but, given a word w, we construct a graph G which contains a wall or a labeled tree-ordered web encoding w with power k. The rest follows then as before.

This concludes the proof of Theorem 8.3. It is easily seen that the proof can be adapted to classes of graphs closed under spanning subgraphs, i.e. edge deletion: instead of taking subgraphs we simply delete all edges no longer needed and make the MSO_2-formulas ignore isolated vertices.

Corollary 8.13. *Let $\mathcal{C}$ be a class of graphs closed under spanning subgraphs, i.e. $G \in \mathcal{C}$, $V(H) = V(G)$, and $E(H) \subseteq E(G)$ implies $H \in \mathcal{C}$.*

1. *If the treewidth of $\mathcal{C}$ is strongly unbounded by $\log^{28^\gamma} n$, where $\gamma > 1$ is larger than the gap-degree of $\mathcal{C}$, then $\mathrm{MC}(\mathrm{MSO}_2, \mathcal{C})$ is not in XP, and hence not fixed-parameter tractable, unless* Sat *can be solved in subexponential time $2^{o(n)}$.*

2. *If the treewidth of $\mathcal{C}$ is strongly unbounded polylogarithmically then $\mathrm{MC}(\mathrm{MSO}_2, \mathcal{C})$ is not in XP unless all problems in the polynomial-time hierarchy can be solved in subexponential time.*

8.4 Conclusion and Outlook

We have presented a strong intractability result for MSO_2 on graph classes of unbounded treewidth. In comparison to Courcelle's theorem, Courcelle's theorem requires the treewidth to be constant whereas our result refers to classes whose treewidth is essentially not bounded logarithmically. As the examples in [MM03] show, there are classes of graphs of unbounded treewidth, closed under subgraphs, which admit tractable MSO_2-model-checking. On the other hand, this is very unlikely to be the case for all classes of logarithmic treewidth. Exploring tractability and intractability of MSO_2 on classes of unbounded treewidth, but bounded by $\log n$, might lead to interesting new results on the boundary of MSO_2-tractability.

The results reported in this part of the thesis refer to MSO_2, i.e. MSO with quantification over sets of edges. For MSO without edge set quantification, referred to as MSO_1, it can be shown that MSO_1 is tractable on any class $\mathcal{C}$ of graphs of bounded *cliquewidth*. Again, except for the examples in [MM03], not much is known about MSO_1 and graph classes of unbounded cliquewidth and it would be very interesting to establish similar results as in this work for the case of cliquewidth. This, however, is much more difficult as no good obstruction similar to grid-like minors is known for this case.

A Preliminaries

We review some basic notions of graph theory, complexity theory, parameterized complexity, and logic. Our graph theoretic notions closely follow that of the books of Diestel [Die05] and Mohar and Thomassen [MT01]. We refer to these books for further background and more detailed explanations of the concepts. We let $\mathbb{N}$ denote the set of positive integers, $\mathbb{N}_0$ the set of nonnegative integers, $\mathbb{Z}$ the set of all integers, $\mathbb{Q}$ the set of rational numbers, and $\mathbb{R}$ the set of real numbers. For integers $n \leq m$, we write $[n, m]$ for $\{n, n+1, \ldots, m\}$ and let $[n] := [1, n]$.

A.1 Basic Graph-Theoretic Concepts

An undirected graph $G = (V, E)$ is a pair where V is a set of *vertices* and E is a set of 2-element subsets of V called the *edges* of G. We usually denote graphs by letters G, H, and refer to their vertex/edge sets by $V(G)$ and $E(G)$, respectively. The *order* of a graph is defined as its number of vertices; unless mentioned otherwise, our graphs have n vertices and m edges. For an edge $e = \{u, v\}$, we often write $e = uv$ and call u and v the *endpoints* of e; in this case, we say u and v are *adjacent*, u is a *neighbor* of v, e *connects* or *joins* u and v, and u is *incident* to e. If two edges e_1 and e_2 share an endpoint, we say e_1 and e_2 are *adjacent*. The *degree* of a vertex v, denoted by $\deg(v)$, is defined as the number of edges that have v as an endpoint. A graph G is *k-regular* if every vertex of G has degree exactly k.

A graph G' is a *subgraph* of G if and only if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$; in this case we write $G' \subseteq G$. If $V(G) = V(G')$ we call G' a *spanning subgraph* of G. For a subset $U \subseteq V(G)$, we write $G[U]$ to denote the subgraph of G *induced by* U, i.e. the graph that has U as its vertex set and an edge $e = uv$ if and only if uv is also an edge in G. Similarly, for a set of edges $F \subseteq E(G)$, the *edge-induced* subgraph $G[F]$ is the graph that consists of all the vertices that have an endpoint in F together with the edges of F.

Two graph G and H are *isomorphic* if there exists a bijection $f : V(G) \to V(H)$ such that uv is an edge of G if and only if $f(u)f(v)$ is an edge of H. The function f is called an *isomorphism* on graphs. We often write $G = H$ to denote that G and H are isomorphic.

The *union* of a number of graphs $G_1, \ldots G_k$ is defined as the graph G with $V(G) = V(G_1) \cup \cdots \cup V(G_k)$ and $E(G) = E(G_1) \cup \cdots \cup E(G_k)$. We define the *intersection of graphs* analogously. For a set of vertices $U \subseteq V(G)$, we write $G - U$ for the graph $G[V(G) \setminus U]$; similarly, for a set of edges $F \subseteq E(G)$, we write $G - F$ for $G[E(G) \setminus F]$. We often use the short notations $G - v$ and $G - e$ for $G - \{v\}$ and $G - \{e\}$, respectively. The corresponding operations are called *vertex deletion* and *edge deletion*.

A.1.1 Paths, Cycles, Walks, and Tours

An *n-path* is a graph $\mathbf{P}_n$ having $n+1$ distinct vertices $v_1, \ldots, v_{n+1}$ and n edges $e_1, \ldots, e_n$ such that $e_i = v_i v_{i+1}$, for $1 \leq i \leq n$. The vertices v_1 and v_{n+1} are called the *endpoints* of $\mathbf{P}_n$ whereas the vertices $v_2, \ldots, v_n$ are called the *internal vertices* of $\mathbf{P}_n$. An *n-cycle* is a graph $\mathbf{C}_n$ with n distinct vertices $v_1, \ldots, v_n$ and n edges $e_1, \ldots, e_n$ such that $e_i = v_i v_{i+1}$, for $1 \leq i \leq n-1$, and $e_n = v_1 v_n$.

A *path of length k* in a graph G is a subgraph P of G that is isomorphic to $\mathbf{P}_k$. We say P *connects* u and v in G if u and v are the endpoints of P. Two paths are called *disjoint* if their vertex and edge sets are disjoint; they are *internally disjoint* if they are disjoint except possibly for their endpoints; and they are called *edge-disjoint* if their edge-sets are disjoint. A *cycle C of length k* of G is a subgraph of G that is isomorphic to $\mathbf{C}_k$. If there exists an edge uv in G such that u and v are on C but their isomorphic copies in $\mathbf{C}_k$ are not adjacent, we call uv a *chord* of C. If a cycle C is chordless, we call C an *induced cycle*, i.e. C is an induced subgraph of G isomorphic to $\mathbf{C}_k$.

A *walk of length k* in a graph G is a sequence $v_1 e_1 v_2 \ldots e_k v_{k+1}$ of vertices and edges of G such that $e_i = v_i v_{i+1}$ for $1 \leq i \leq k$. We often denote a walk simply by its sequence of vertices, i.e. as $v_1 \ldots v_{k+1}$. A walk is *closed* if $v_1 = v_{k+1}$. Note that a path is a walk that does not repeat vertices and a cycle is a closed walk that has distinct vertices except for $v_1 = v_k$; hence, we may use a similar notation as for walks for paths and cycles as well. Sometimes we speak of a *simple* path or cycle to emphasize the fact that the considered path or cycle consists of distinct vertices.

A *Hamiltonian cycle* of a graph G is a cycle that visits every vertex of G exactly once. An *Euler tour* is a closed walk in G that visits every edge of G exactly once. It is easy to see that a graph contains an Euler tour if and only if it is connected and every vertex is of even degree.

A.1.2 Connectivity and Separation

Two vertices u and v in G are *connected* if there exists a path in G with endpoints u and v. The graph G itself is called *connected* if every pair of its vertices are connected; otherwise it is called *disconnected*. The maximal connected subgraphs of G are called the *connected components* of G.

A set of vertices X in a connected graph G with the property that $G-X$ is disconnected is called a *(vertex) separator* of G; similarly, a set of edges F where $G-F$ is disconnected is called an *(edge) cut* of G. The *size* of a separator or cut is defined as the number of its elements. A single vertex v whose removal disconnects G is called a *cut vertex*; a single edge e whose removal disconnects G is called a *cut edge*. In this thesis, we will primarily work with vertex separators and refer to them simply as separators if no ambiguity arises.

A graph G is called *k-connected* if the order of G is at least $k+1$ and for any set U of at most $k-1$ vertices the graph $G-U$ is connected; in other words, a k-connected graph has no separators with less than k vertices. By Menger's famous theorem [Men27], this is equivalent to the following condition: for any two vertices $u, v \in V(G)$, there exist at

least k internally disjoint paths between u and v. Since this is a standard theorem in graph theory, we may use these characterizations of k-connected graphs interchangeably. Similarly, we say that a graph G is *k-edge-connected* if for any set F of at most $k-1$ edges, $G-F$ is connected. Equivalently, a graph G is k-edge-connected if for any two vertices $u, v \in V(G)$, there exist at least k *edge-disjoint* paths between u and v in G.

By the definition above, we have that a graph is 2-connected, or *biconnected*, if and only if it does not have a cut vertex; equivalently, for any two vertices u, v in a biconnected graph, there exists a cycle that contains both u and v. If we consider the maximal biconnected subgraphs of a given graph, we obtain a partition of the edge set of the graph into its *biconnected components*, which are also called *blocks*. Note that every cut edge induces a block of its own and that two blocks meet only at a cut vertex.

A.1.3 Forests and Trees

A *forest* is a graph that contains no cycles. A *tree* is a connected forest. The vertices of a tree are sometimes also called *nodes*. The vertices of degree 1 in a tree T are called *leaves*; the vertices of higher degree are called *internal vertices* or *branching vertices*. Every tree with at least two vertices contains at least two leaves. A tree on n vertices has exactly $n-1$ edges. For any two vertices u and v in a tree T, there exists a unique path in T that connects u and v. Every edge of a tree is a cut edge and every internal vertex is a cut vertex. Every connected graph G contains at least one *spanning tree* T, i.e. a spanning subgraph that is a tree.

Oftentimes trees are considered to have a distinguished vertex r called the *root* of the tree defining a certain hierarchy on the tree. For every vertex v other than the root, its *parent* is defined as the unique vertex u that precedes v on the path from r to v. Conversely, v is said to be a *child* of u. The notion of a *sibling* is defined as naturally. More generally, the vertices that appear on the path from r to v are called the *ancestors* of v in the tree.

An *Euler tour of a tree* T is a closed walk in T that traverses each edge *exactly once in each direction*; i.e. every edge is traversed exactly *twice*. Every tree contains an Euler tour that can be obtained simply by walking "along its boundary" (cf. Figure 1.4 (a)-(b)).

A.1.4 Cliques, Independent Sets, Bipartite Graphs

The *complete graph* $\mathbf{K}_n$ is defined as the graph on n vertices in which every two vertices are adjacent. A *clique* in a graph G is a subgraph of G that is isomorphic to some complete graph. Sometimes we call a complete graph also a clique. On the other hand, a set of vertices U in a graph G is called *independent* if no two vertices of U are adjacent.

A graph G is called *bipartite* if its vertex set can be partitioned into 2 sets U and W such that each of them is an independent sets, i.e. every edge of G has an endpoint in U and an endpoint in W. It is well known that a graph is bipartite if and only if it contains no odd cycle. A *complete bipartite graph* $\mathbf{K}_{m,n}$ is a bipartite graphs with parts U and W of size n and m, respectively, such that there is an edge between every vertex of U and every vertex of W.

More generally, a graph G is called *k-partite* if its vertex set can be partitioned into k parts such that each part is an independent set. A *complete k-partite graph* is a k-partite graph that includes every allowed edge.

A.1.5 Matchings, Covers, Domination

A *matching* M in a graph G is a subset of the edges of G such that no two edges in M are adjacent. A *maximal matching* is a matching that cannot be extended by another edge whereas a *maximum matching* is a matching of maximum possible cardinality. A *vertex cover* of a graph G is a subset U of the vertices of G such that every edge of G has an endpoint in U. By Königs Theorem [Kön31], we know that in every bipartite graph, the size of a maximum matching is equal to the size of a minimum vertex cover. However, this is not true in general graphs.

A *dominating set* in a graph G is a set $D \subseteq V(G)$ such that every vertex of G is either in D or has a neighbor in D. An *edge dominating set* is a set of edges $E_D \subseteq E(G)$ such that every edge not in E_D is adjacent to an edge in E_D.

A.1.6 Weighted Graphs, Directed Graphs, Multigraphs

We often consider *edge-weighted* graphs: a graph G together with a function $\ell : E(G) \to \mathbb{R}$ that assigns a weight to every edge of G. Unless otherwise mentioned, we restrict ourselves to nonnegative integer weights, i.e. weight functions of the form $\ell : E(G) \to \mathbb{N}_0$. We define the weight of an edge set $F \subseteq E(G)$ as the sum of the weights of the edges in it and denote it by $\ell(F)$; the weight of a subgraph $H \subseteq G$ is defined as $\ell(H) := \ell(E(H))$. We occasionally use the term *length* to refer to the weight of an edge or subgraph.

We might sometimes also consider *vertex-weighted* graphs, in which case we will make it explicit to avoid confusion. A graph with weights on its vertices is associated with a function $w : V(G) \to \mathbb{R}$ that assigns a weight to each vertex of the graph. We generalize the function w to sets of vertices and subgraphs as in the case of edge weights.

In Chapter 6, and occasionally at some other places, we consider some problems on *directed graphs*. A directed graph $D = (V, E)$ is a pair consisting of a set of vertices V and a binary relation E on V called the directed edges of D. An edge $e = (u, v) \in E$, sometimes denoted as uv is said to be *directed* from u to v. Directed edges are usually drawn as arrows showing the direction of the edge. Like in the case of undirected graphs, we refer to the set of vertices of a directed graph D by $V(D)$ and to its set of edges by $E(D)$. One can regard undirected graphs as directed graphs in which the edge relation is symmetric. Most of the notions on undirected graphs translate seamlessly to directed graphs, so we will only define specific notions on directed graphs when necessary.

A *multigraph* is a graph that additionally might have loops and parallel edges. A *loop* is an edge that has the same vertex as both of its endpoints; *parallel edges* are multiple edges that all have the same endpoints. One can think of each edge and the edge set of a multigraph as multisets. Formally, a multigraph is a triple $G = (V, E, \varrho)$, where V is the set of vertices of G, E is its set of edges, and ϱ is a function that assigns to each edge in E a pair of vertices from $V \times V$ as its endpoints. Most notions from

graphs adapt seamlessly to multigraphs and hence we will only define them explicitly if necessary. Multigraphs are sometimes considered in conjunction with embedded graphs in our work.

A.1.7 Graph Traversals

A *graph traversal* is an algorithm that visits every vertex and every edge of a graph in a specific order. The most prominent graph traversal algorithms are *breadth-first search (BFS)* and *depth-first search (DFS)*. In BFS, we start at some root vertex r, mark it as visited, and put it in a queue. At every step, we look at the vertex at the head of the queue, remove it, consider all its unvisited neighbors, mark them as visited, and put them at the end of the queue. Additionally, we may assign every vertex of the graph to a *BFS-layer* when performing this procedure: the root is defined to be on layer zero and whenever the neighbors of a vertex on some layer i are inserted into the queue, they are assigned to layer $i + 1$. This layering turns out to be very useful in many algorithms.

In DFS, the same procedure is executed using a stack instead of a queue; though, it is usually described and implemented recursively instead of explicitly mentioning a stack data structure. One further differentiates between *pre-order*, *in-order*, and *post-order* DFS traversals depending on whether the current vertex is visited, i.e. processed, before, in-between, or after its children. In-order traversals are often used in conjunction with *binary trees*, rooted trees in which every vertex has at most two children, especially when there is a notion of a left child and a right child at each vertex. The order in which vertices are visited in a DFS procedure induces a numbering on the vertices called a *DFS-numbering*; note however, that there exist different DFS-numberings based on using a pre-order, in-order, or post-order transversal and also depending on the order in which the children of a vertex are visited. Many graph algorithms can be described succinctly based on a BFS or DFS procedure by basically specifying only the steps that have to be performed upon visiting a vertex.

A.1.8 Distances and Neighborhoods

The *length* of a path P in an unweighted graph G is defined as the number of edges of P; in a weighted graph, it is defined as the sum of the weights of the edges on the path. A *shortest path* between two vertices u and v is a path of minimum length between u and v. The *distance* between u and v in G, denoted by $\text{dist}_G(u, v)$, is defined as the length of a shortest path between u and v; we might omit the subscript G whenever it is clear from context. In a weighted graph, we use the terms *unweighted length* and *unweighted distance* when considering the number of edges of paths instead of their weight. Note that if we run a BFS rooted at a vertex u, the BFS-layer of each vertex v equals its unweighted distance from u. The weighted distance of vertices can be calculated efficiently in graphs with nonnegative weights using Dijkstra's algorithm [Dij59].

The *r-neighborhood* of a vertex v, denoted by $N_r(v)$, is the set of vertices at unweighted distance at most r from v; we define $N(v) = N_1(v)$. Note that by our definition, v itself is included in its neighborhoods.

A.2 Embeddings, Minors, and Treewidth

A *drawing* of a graph G in the plane is obtained by assigning every vertex of G to a point in the plane and every edge of G to a simple (polygonal) curve in the plane such that the endpoints of the curve correspond to the positions of the endpoints of the edge; the curves of distinct edges may generally cross each other but their interiors may not contain any vertex. It is convenient to sometimes identify a graph and its drawing. A *planar graph* is a graph that can be drawn in the plane in such a way that distinct edges do not cross each other; in this case, we say that the graph is *embedded in the plane*. An embedding of a planar graph partitions the plane into a number of regions, each of which is called a *face* of the embedding or the graph; furthermore, there always exists one distinguished *infinite* or *outer face* of the embedding. Note that by a simple geometric transformation, an embedding can be changed in such a way that any particular face becomes the outer face. If n_f denotes the number of faces of an embedding of a planar graph, we have by Euler's formula that $m = n + n_f - 2$; hence, the number of faces of a planar graph does not depend on a particular embedding.

A.2.1 Embeddings of Graphs on Surfaces

We study embeddings on more general surfaces, such as a torus or the projective plane, as outlined in [MT01] and refer to this book for a more detailed explanation of the concepts presented in this section. We consider only compact surfaces without boundary. If a surface contains a subset homeomorphic to a Möbius strip, it is *nonorientable*; otherwise it is *orientable*. In this thesis, an embedding refers to a *2-cell embedding*, i.e. a drawing of the vertices and edges of the graph as points and arcs on a surface such that every face is homeomorphic to an open disc. Such an embedding can be described purely combinatorially by the notion of a *combinatorial embedding*:

Let G be a connected multigraph. A *rotation system* is a set of permutations $\{\pi_v \mid v \in V(G)\}$ that we think of as specifying the cyclic ordering of the edges around each vertex in clockwise order. A *combinatorial embedding* is a pair $\Pi = (\pi, \lambda)$ where π is a rotation system and $\lambda : E(G) \to \{-1, 1\}$ is a *signature mapping*. For an edge $e \in E(G)$ incident to a vertex $v \in V(G)$, we call the cyclic sequence $e, \pi_v(e), \pi_v^2(e), \ldots$ the Π-*clockwise ordering* around v. Given an embedding Π of G, we say that G is Π-*embedded*.

The Π-*facial walks* of G are obtained by the following *face traversal procedure*. Such a traversal can be in clockwise mode or anti-clockwise mode; we start in the clockwise mode. Starting with an arbitrary vertex v and an edge $e_0 = vu$ incident to v, we first traverse the edge e_0 from v to u. If we are in clockwise mode, we continue the walk along the edge $e_1 = \pi_u(e_0)$; if we are in anticlockwise mode we traverse the edge $e_1' = \pi_u^{-1}(e_0)$. Whenever we traverse an edge that has signature -1 we change from clockwise mode to anticlockwise mode or vice versa. We continue this procedure until we get to our starting edge e_0 in the same direction and in the same mode. The other Π-facial walks are determined in the same way by starting with other edges and directions; facial walks that differ only by a cyclic shift are considered to be the same. If $G = K_1$, we define the walk of length 0 to be the facial walk. Two embeddings are equivalent if and only if

they contain the same set of facial walks. One can show that every edge appears exactly twice in the set of all Π-facial walks of G. In fact, every edge appears either twice on the same facial walk or in exactly two facial walks. The edges that are contained twice in one facial walk are called *singular*. Similarly, if v is a vertex that appears more than once on a facial walk, we call it *singular*. We sometimes refer to singular parts of a facial walk as *tree-like parts*. If a facial walk does not contain singular vertices or edges, it is a Π-*facial cycle*. A cycle C of a Π-embedded graph G is called Π-*onesided* if it has an odd number of edges with negative sign; otherwise it is called Π-twosided. If G contains a Π-onesided cycle, the embedding is *nonorientable*; otherwise it is *orientable*.

A combinatorial embedding of a graph G naturally induces a geometric 2-cell embedding (on each subgraph) of G and vice versa. Note that a combinatorial embedding is orientable if and only if it induces a 2-cell embedding on an orientable surface. We often use this correspondence implicitly and talk about the combinatorial and geometric embedding of a graph simultaneously. Note that each facial walk of a combinatorial embedding induces a face in the geometric embedding; and, of course, every face of a geometric embedding corresponds to a unique facial walk. Hence, talking about the faces of an (combinatorial) embedding is one example where we implicitly go from combinatorial to geometric embeddings. For a face f of an embedded graph, we let ∂f denote its facial walk which we sometimes also call *boundary*. When we refer to a planar embedding, we actually mean an embedding in the 2-sphere; in this setting, any face of the planar embedding can be thought of as the "outer face". For an embedded graph G with n_f faces, the number $g = 2 + m - n - n_f$ is called the *Euler genus* of the surface. The Euler genus is equal to twice the usual genus for orientable surfaces and equals the usual genus for nonorientable surfaces.

Let $G = (V, E, \varrho)$ be a multigraph embedded on a surface and $\mathcal{F}$ its set of faces. The *dual* $G^\star = (\mathcal{F}, E^\star, \varrho^\star)$ of G is defined as the multigraph having the set of faces of G as its vertex set and the following set of edges: for every edge $e \in E$ of G that is adjacent to the faces f_1 and f_2 in the embedding of G, there exists an edge $e^\star$ that connects the vertices f_1 and f_2 in $G^\star$. We actually identify e and $e^\star$, set $E^\star := E$, and define G and $G^\star$ to have the same set of edges; only the functions ϱ and $\varrho^\star$ make the distinction. Note that the dual of an embedded graph is in general a multigraph embedded on the same surface.

A.2.2 Contraction and Minors

For an edge $e = uv$ in a graph G, we define the operation G/e of *contracting* e as identifying u and v and removing all loops and parallel edges. For a set of edges $F \subseteq E(G)$, we define the operation G/F as consecutively contracting all edges of F; the order of the contractions is irrelevant. The inverse operation of contraction is that of *splitting* a vertex w, that is, replacing it by an edge uv and assigning the edges that ended in w to u or v. However this operation is not well defined, as it is not clear how to assign these edges; if this operation is needed at any point, this assignment has to be specified.

A graph H is a *minor* of G if it can be obtained from G by a series of vertex and edge deletions and contractions; we write $H \preceq G$ to denote this relation. As $(G - F_1)/F_2 =$

$(G/F_2) - F_1$ for edge sets $F_1, F_2 \subseteq E(G)$, the order of the contractions and deletions does not matter. A class of graphs $\mathcal{C}$ is said to be *minor-closed* or *closed under building minors* if for every graph $G \in \mathcal{C}$, all of its minors also belong to $\mathcal{C}$ (up to isomorphisms). Such a class is called *proper* if it is neither the empty class nor the class of all graphs. For any proper minor-closed class of graphs $\mathcal{C}$, there exists at least one graph H that is excluded from $\mathcal{C}$; the graphs in $\mathcal{C}$ are hence *H-minor-free*. The term *H-minor-free graphs* more generally describes classes of graphs that exclude a fixed graph H as a minor. By a well-known theorem of Mader [Mad67], we know that H-minor-free graphs have *bounded average degree*, i.e. they fulfill $m \leq c_H n$, for a constant c_H depending only on $|H|$. Hence their size is determined basically by their number of vertices.

A *model* of H in G is a map that assigns to every vertex v of H a connected subgraph Z_v of G such that for $u \neq v \in V(H)$, Z_v and Z_u are disjoint and for every edge $uv \in E(H)$, there exists an edge in G that has one endpoint in Z_v and one endpoint in Z_u. W.l.o.g. we may assume that the Z_v are trees as we can take a spanning tree of Z_v otherwise. It is easy to see that a graph H is a minor of G if and only if G contains a model of H.

The operation of *splitting an edge uv* is defined as adding a vertex "in its middle", i.e. replacing it by a path uwv, where w is a new vertex added to the graph. A *subdivision* S of a graph H is a graph that is obtained from H by iteratively splitting some edges. The original vertices of H are sometimes called the *nails* of H in S. H is a *topological minor* of G if a subdivision of H is isomorphic to a subgraph of G. The *nails* of H in G are the nails of the subdivision of H in G. A topological minor H of G is also a minor of G but the reverse is guaranteed to be true only if H has maximum degree 3 [Die05, Proposition 1.7.2].

We also sometimes consider *surface contractions* and *surface minors* of embedded graphs; in this setting, we do *not* delete loops and parallel edges after contracting an edge; furthermore, the contraction of a loop is not allowed. As a consequence, the genus of a graph is preserved when performing only surface contractions on its embedding. Note that a combinatorial embedding of a graph G induces such an embedding for all surface minors of G.

A.2.3 The Graph Minor Theorem

Note that the class of graphs that can be embedded on some fixed surface S is closed under building minors; indeed, deleting and contracting edges preserves an embedding on the surface. Hence, these classes of graphs are some prominent examples of proper minor-closed graph classes. Further classes include, for example, linklessly embeddable graphs, knotlessly embeddable graphs, graphs that have a vertex cover of size at most k (for some constant k), and graphs embedded in some fixed surface augmented by a constant number of additional vertices, called *apices*, that may have edges to arbitrary vertices of the graph.

By Kuratowski's Theorem [Kur30], we know that a graph is planar if and only if it excludes $\mathbf{K}_5$ and $\mathbf{K}_{3,3}$ as a (topological) minor. Similar theorems hold for all fixed surfaces and in fact, for all minor-closed graph classes as is shown by the seminal *Graph*

Minor Theorem of Robertson and Seymour [RS04]: any proper minor-closed class of graphs is characterized by a *finite* number of excluded minors. This is one of the deepest and most far-reaching results of graph theory in the recent decades and has been proven in a course of over 20 papers. We will return to this theorem and the graph minor theory in Parts II and III of the thesis.

A.2.4 Tree Decompositions and Treewidth

A *tree decomposition* of a graph G is a pair $(T, \mathcal{B})$, where T is a tree and $\mathcal{B} = \{B_i | i \in V(T)\}$ is a family of subsets of $V(G)$, called *bags*, such that

(i) every vertex of G appears in some bag of $\mathcal{B}$;
(ii) for every edge $e = uv$ of G, there exists a bag that contains both u and v; and
(iii) for every vertex v of G, the set of bags that contain v form a connected subtree T_v of T.

The *width* of a tree decomposition is the maximum size of a bag in $\mathcal{B}$ minus 1. The *treewidth* of a graph G, denoted by $\mathrm{tw}(G)$, is the minimum width over all possible tree decompositions of G. One can think of the treewidth of a graph as a global connectivity measure that describes how similar the given graph is to a tree. Indeed, the class of graphs of treewidth 1 is exactly the class of all forests; the class of graphs of treewidth 2 is the class of series-parallel graphs.

Let $f : \mathbb{N}_0 \to \mathbb{N}_0$ be a function and $\mathcal{C}$ be a class of graphs. The treewidth of $\mathcal{C}$ is *bounded by* f if $\mathrm{tw}(G) \leq f(|G|)$ for all $G \in \mathcal{C}$. $\mathcal{C}$ has *bounded treewidth* if its treewidth is bounded by a constant.

The notion of treewidth has had an enormous impact on both theory and practice in computer science since many computationally hard problems become easy on graphs of bounded treewidth, see e.g. [AP89, Bod88]. Indeed, Courcelle's famous theorem [Cou90] tells that any problem that can be formulated in monadic second-order logic can be decided on graphs of bounded treewidth in linear time. Furthermore, the notions of tree decomposition and treewidth turn out to be of the most useful and important tools in theoretical computer science and graph theory, especially in graph minor theory and parameterized complexity.

A.2.5 On Local Treewidth and Apex-Minor-Free Graphs

We say that a graph has *bounded local treewidth* if for every vertex v and integer r, we have $\mathrm{tw}(N_r(v)) \leq f(r)$, for some computable function f depending solely on r; we write $\mathrm{ltw}_r(G) \leq f(r)$. Demaine and Hajiaghayi [DH04] showed that every minor-closed class of graphs that has bounded local treewidth has, in fact, *linear* local treewidth, i.e. in this case, we have $\mathrm{ltw}_r(G) \leq \lambda r$ for a fixed integer λ depending only on the excluded minor of the class.

An *apex-graph* is a planar graph augmented by an additional vertex that can have edges to any other vertex. A class of graphs is called *apex-minor-free* if it excludes a fixed apex-graph as a minor. Eppstein [Epp00a] showed that a minor-closed class of graphs has bounded local treewidth if and only if it is apex-minor-free.

A.3 On Classical and Parameterized Complexity

We assume basic familiarity with fundamental notions of complexity as given in the book by Garey and Johnson [GJ79]. We define the class P to be the class of problems (i.e. languages) that can be decided by a deterministic Turing machine in polynomial time and NP the class of problems that can be decided by a nondeterministic Turing machine in polynomial time. A polynomial-time (many-one) reduction from a problem A to a problem B, denoted as $A \leq^p B$, is a polynomial-time computable function $f : A \to B$ such that $x \in A \Leftrightarrow f(x) \in B$. A problem A is said to be NP-*hard* if for every problem $B \in \mathsf{NP}$, we have $B \leq^p A$; if additionally A is itself in NP, it is called NP-*complete*.

A.3.1 Landau Notation

We use the standard Landau (big-Oh) notation: for a function $f : \mathbb{N}_0 \to \mathbb{N}_0$, we let

$$\begin{aligned}
\mathcal{O}(f) &:= \{g : \mathbb{N}_0 \to \mathbb{N}_0 \mid \exists c, n_0 \in \mathbb{N}_0 \, \forall n \geq n_0 : g(n) \leq c \cdot f(n)\} \\
\Omega(f) &:= \{g : \mathbb{N}_0 \to \mathbb{N}_0 \mid \exists c, n_0 \in \mathbb{N}_0 \, \forall n \geq n_0 : g(n) \geq \frac{1}{c} \cdot f(n)\} \\
\Theta(f) &:= \mathcal{O}(f) \cap \Omega(f) \\
o(f) &:= \{g : \mathbb{N}_0 \to \mathbb{N}_0 \mid \forall c \in \mathbb{N}_0 \, \exists n_0 \in \mathbb{N}_0 \, \forall n \geq n_0 : g(n) \leq \frac{1}{c} \cdot f(n)\} \\
\omega(f) &:= \{g : \mathbb{N}_0 \to \mathbb{N}_0 \mid \forall c \in \mathbb{N}_0 \, \exists n_0 \in \mathbb{N}_0 \, \forall n \geq n_0 : g(n) \geq c \cdot f(n)\}
\end{aligned}$$

As usual, we often write $f(n) = \mathcal{O}(g(n))$ or $f(n) \leq \mathcal{O}(g(n))$ for $f(n) \in \mathcal{O}(g(n))$. When writing $\mathcal{O}(1)$, we regard 1 as the constant function with value 1. If we want to emphasize that the hidden constants in such an expression depend on some parameter k, we use the notation $\mathcal{O}_k(f)$ (where we always assume that the dependence is at least computable); this is especially common with regard to H-minor-free graphs where we use the notation $\mathcal{O}_H(f)$ to emphasize that the hidden constants depend on the excluded minor H. We also use some derived notation such as

$$n^{\mathcal{O}(f(n))} := \{g : \mathbb{N}_0 \to \mathbb{N}_0 \mid \exists f' \in \mathcal{O}(f) \, \exists n_0 \in \mathbb{N}_0 \, \forall n \geq n_0 : g(n) \leq n^{f'(n)}\}.$$

In particular, the class $\mathrm{poly}(n) := n^{\mathcal{O}(1)}$ is the class of all polynomially bounded functions.

A.3.2 On Parameterized Complexity

We use the standard notions of parameterized complexity as given in the books of Downey and Fellows [DF99] and Flum and Grohe [FG06]. A *parameterized problem* is a pair (Q, κ) where Q is a problem (i.e. a language over some alphabet) and κ is a *parameterization*, that is, a polynomial-time computable function that assigns an integer to every instance of Q (formally, any word from the alphabet of Q is an instance). For an instance x of Q, we call the number $\kappa(x)$ the corresponding *parameter*. For example,

in k-Independent Set an instance consists of a graph G and an integer k and the parameter is defined to be k; the question is whether the graph G contains an independent set of size at least k. This is a typical example of the *standard parameterization* of a problem where the parameter is chosen to be the solution size. Other examples of typical parameters are the treewidth of the instance, the genus of the given graph, and the size of an excluded minor.

The Classes FPT and XP A parameterized problem is said to be *fixed-parameter tractable* (FPT) if for any instance of size n with parameter k it can be solved in time $\mathcal{O}_k(n^{\mathcal{O}(1)}) = \mathcal{O}(f(k)n^{\mathcal{O}(1)})$, for some computable function f solely dependent on k; a corresponding algorithm is called an FPT-*algorithm*. We denote the class of parameterized problems that are fixed-parameter tractable by FPT. We define the class XP to contain all parameterized problems that can be solved in time $\mathcal{O}(n^{f(k)})$ for some computable function f. Clearly, FPT $\subseteq$ XP; in fact, this inclusion is strict [DF99]. Whereas many parameterized problems admit simple XP-algorithms, showing the fixed-parameter tractability of problems is often hard and builds one of the main focuses of parameterized complexity.

The W-hierarchy Another main focus of the theory is to provide evidence that many parameterized problems most likely do *not* admit an FPT-algorithm. The complexity classes W[1] $\subseteq$ W[2] $\subseteq \cdots \subseteq$ W[P] are defined for this purpose. The assumption that FPT $\neq$ W[1] is the parameterized analog of the assumption that P $\neq$ NP in classical complexity. We refer to the books mentioned above [DF99, FG06] for formal definitions and further background on the W-hierarchy.

Bounded Fixed-Parameter Tractability Even after a problem is shown to be in FPT, it remains a challenge to reduce the dependence on k and find the smallest function f for which the problem still admits an algorithm with running time $\mathcal{O}(f(k)n^{\mathcal{O}(1)})$. To this end, the notion of *bounded* FPT is introduced. If the function f above belongs to a class $\mathcal{F}$ of functions from $\mathbb{N}$ to $\mathbb{N}$, we say that our problem is in $\mathcal{F}$-FPT. We denote $2^{k^{\mathcal{O}(1)}}$-FPT, $2^{\mathcal{O}(k)}$-FPT, and $2^{o^{\text{eff}}(k)}$-FPT by EXPT, EPT, and SUBEPT, respectively; here, $f \in o^{\text{eff}}(g)$ if there exists $n_0 \in \mathbb{N}$ and a computable, nondecreasing and unbounded function $\iota : \mathbb{N} \to \mathbb{N}$, such that $f(n) \leq \frac{g(n)}{\iota(n)}$ for all $n \geq n_0$. A problem is *subexponential fixed-parameter tractable* if it is in SUBEPT.

Kernels A *kernel* of a parameterized problem is a fully polynomial algorithm that given an instance of size n and parameter k, returns a *equivalent reduced instance* of the same problem of size $f(k)$ and parameter $k' \leq k$. The function f denotes the *size* of the kernel; we then speak of a *linear, polynomial, and subexponential* kernel, respectively. Kernelization is a major technique in fixed parameter complexity as any computable parameterized problem is in FPT if and only if it admits a kernel [Nie02]. Kernelization can be seen as polynomial-time pre-processing with a quality guarantee and has gained much theoretical importance in the recent years – besides its natural

practical importance. For an introduction to kernels we refer to the survey by Guo and Niedermeier [GN07].

Exponential Time Hypothesis The *satisfiability problem* (SAT) is that of deciding if a given CNF-formula is satisfiable; k-SAT is the variant where each clause is restricted to have at most k variables. SAT was the first problem shown to be NP-complete [Coo71] and 3-SAT is well-known to be NP-complete either [Kar72]. The *exponential time hypothesis* (ETH) is the assumption that 3-SAT can not be solved in time $2^{o^{\text{eff}}(n)}$ where n is the number of variables of the given instance. This is equivalent to saying that 3-SAT parameterized by the number of variables is not in SUBEPT. This assumption is widely believed and is implied by the stronger assumption that FPT $\neq$ W[1]. Many negative results in parameterized complexity are based on this assumption.

A.3.3 Optimization Problems and Approximation

An NP-*optimization problem* is a tuple (I, S, C, opt) where I is a polynomial-time decidable set of *instances*, S is a mapping that associates a nonempty set of *solutions* $S(x)$ with each instance $x \in I$ such that for a given pair (x, y), it is decidable in polynomial time whether $y \in S(x)$ and there exists a $k \in \mathbb{N}$ such that for all $x \in I$ and $y \in S(x)$, we have $|y| \leq |x|^k$; furthermore, $C : \{(x, y) \mid x \in I, y \in S(x)\} \to \mathbb{N}$ is a polynomial-time computable *cost function* and $\text{opt} \in \{\min, \max\}$ is the *optimization goal*.

Given a NP-optimization problem P and an instance $x \in I$, we want to find a $y \in S(x)$ such that

$$C((x, y)) = \text{opt}(x) = \text{opt}\{C(x, z) \mid z \in S(x)\}\,.$$

We sometimes denote this optimal value with $\text{OPT}_{P,x}$ and drop the indices if the problem and/or instance at hand is clear from context. We denote the class of all NP-optimization problems by NPO. In what follows, let $P = (I, S, C, \text{opt})$ be an NP-optimization problem.

Let $x \in I$ be an instance of P and $y \in S(x)$ a solution. We say y is an α-*approximation* for x if $\text{ratio}(x, y) := \max\{\frac{C((x,y))}{\text{OPT}}, \frac{\text{OPT}}{C((x,y))}\} \leq \alpha$. Note that by our definition, we have $\alpha \geq 1$ for both maximization and minimization problems. Let $\mathcal{A}$ be a polynomial-time algorithm that for a given instance $x \in I$ computes a solution $y \in S(x)$ and $r : \mathbb{N} \to \mathbb{N}$ a computable function. We say $\mathcal{A}$ is an r-*approximation algorithm* for P if we have $\text{ratio}(x, y) \leq r(|x|)$ for all instances $x \in I$ and solutions $y \in S(x)$ computed by $\mathcal{A}$. We often call the function r the *approximation ratio* or *guarantee* of $\mathcal{A}$. If, in particular, r is a constant function, we say $\mathcal{A}$ is a *constant-factor approximation algorithm* for P. We denote the class of all NP-optimization problems that admit a constant-factor approximation by APX. We call a problem Q APX-hard if every problem in APX can be reduced to Q by a polynomial-time approximation-preserving reduction; Q is APX-complete if it is APX-hard and in APX. We refer to the book of Ausiello et al. [ACG+03] for further details (e.g. the definition of approximation-preserving reductions).

A.3.4 Polynomial-Time Approximation Schemes

A solution y for an instance x of an NP-optimization problem P is called ε-close if $\text{ratio}(x, y) \leq (1+\varepsilon)$. We often call such a solution also *near optimal*. A *polynomial-time approximation scheme* (PTAS) for P is a family of algorithms $(\mathcal{A}_\varepsilon)_{\varepsilon \geq 0}$, so that $\mathcal{A}_\varepsilon$ is an $(1+\varepsilon)$-approximation algorithm for P, i.e. $\mathcal{A}_\varepsilon$ runs in polynomial time in $|x|$ for any instance x of P and computes ε-close solutions. A crucial property that we require from PTASes is that the family $(\mathcal{A}_\varepsilon)_{\varepsilon \geq 0}$ be *uniform*, in the sense that there exists a polynomial-time algorithm that for a given $0 \leq \varepsilon \in \mathbb{Q}$ computes $\mathcal{A}_\varepsilon$. We use the notion PTAS also to denote the class of all NP-optimization problems that admit a PTAS. We trivially have PTAS $\subseteq$ APX $\subseteq$ NPO. It is well-known that unless P = NP, all these inclusions are *strict*; this implies, in particular, that APX-hard problems most likely *do not* admit a PTAS.

Note that the running time of a PTAS is, in fact, $\mathcal{O}_\varepsilon(\text{poly}(n))$ for an instance of size n; that is, in general, a PTAS runs in polynomial time only for a *fixed error bound*. Hence, there is a qualitative difference in the running times of different PTASes based on their dependence on $1/\varepsilon$ that may become very important, especially when it comes to their practical applicability; this difference is best studied in the framework of parameterized complexity theory: a PTAS can naturally be seen as an algorithm parameterized by $1/\varepsilon$. Whereas many PTASes are in XP in this respect, there have been great efforts in recent years in obtaining *efficient* PTAS*es* (EPTASes), namely, PTASes that are in FPT with respect to the parameter $1/\varepsilon$, see e.g. [Bak94, RS98, Gro03, Kle08]. Of course, even after an EPTAS or an FPT-algorithm is discovered for a problem, it still remains to find the algorithm with the smallest dependence on the parameter; indeed, we will study cases in this thesis where such an improvement on the parameter-dependence turns out to be crucial (see Chapters 3 and 6).

If the dependence on $1/\varepsilon$ of an EPTAS is even polynomial, we obtain a *fully polynomial-time approximation scheme* (FPTAS); problems admitting an FPTAS define the complexity class FPTAS. However, no polynomially bounded NP-optimization problem (i.e. a problem with $C((x, y)) \leq |x|^{\mathcal{O}(1)}$ for all $x \in I$ and $y \in S(x)$) admits an FPTAS unless P = NP [KS81]. We trivially have FPTAS $\subseteq$ EPTAS $\subseteq$ PTAS; Marx [Mar07, Mar08] showed that unless ETH fails, all these inclusions are *strict*.

A.3.5 Some Important NP-Hard Problems

Finally, we review some of the most important (NP-hard) problems we consider and their basic properties. In Vertex Cover, we are given a graph G and an integer k and are asked if G contains a vertex cover of size at most k; in the optimization version we are asked for a vertex cover of minimum size. The problem is APX-complete [PY91], admits only a 2-approximation in general graphs (see [Hal02] for slight nonconstant improvements) but a PTAS on H-minor-free graphs [Gro03]. If parameterized by k, we obtain k-Vertex Cover, which is in EPT in general graphs [DF95c] and in SUBEPT on H-minor-free graphs [DFHT05].

The Independent Set problem is defined as the set of pairs (G, k) such that G is a

graph that contains an independent set of size at least k; similarly Clique is the set of such pairs where G contains a clique of size at least k. Their optimization versions are naturally maximization problems. Both problems are APX-hard [PY91] and the best known approximation algorithms have a ratio of $\mathcal{O}(n/\log^2 n)$ [BH92]; but they admit a PTAS on H-minor-free graphs [Gro03]. They are W[1]-complete when parameterized by k [DF95b] but in SUBEPT on H-minor-free graphs [DFHT05].

In Dominating Set we are asked if a given graph G contains a dominating set of size at most k (or for a dominating set of smallest size in the optimization version). The problem is APX-hard in general graphs [PY91] with the best known approximation ratio being $1 + \log n$ [Joh74]. It admits a PTAS on H-minor-free graphs [Gro03]. The parameterization by k is k-Dominating Set and is W[2]-complete in general graphs [DF95a] but in SUBEPT on H-minor-free graphs [DFHT05].

We also sometimes consider the *connected* or *independent* version of the problems above where we require the solution sets to be connected or independent. These variants are usually harder than the unconstrained versions; we provide more details when required.

Max-Cut refers to the optimization problem of finding a cut with the maximum number of edges in a given graph. Whereas the Min-Cut problem is well-known to be solvable in polynomial time, the Max-Cut problem is indeed NP-hard. It admits a constant-factor approximation of 1.1383 [GW95] but is APX-complete in general graphs [PY91]. It admits a PTAS on dense graphs, i.e. when $m = \Theta(n^2)$ [AKK99] and on H-minor-free graphs [DHK05]. The parameterized version k-Max-Cut, where we are asked for a cut of size at least k in a given graph, is in EPT in general graphs [RS07] and trivially in SUBEPT on H-minor-free graphs (every graph has a cut of size at least $m/2$ and H-minor-free graphs have treewidth $\mathcal{O}(\sqrt{n})$ [AST90]).

The Hamiltonian Path and Hamiltonian Cycle problems are that of deciding whether a given graph contains a Hamiltonian path or cycle, respectively. Their optimization version is commonly referred to as the *traveling salesman problem* (Tsp); in the general version, we are given a (possibly edge-weighted) graph and are asked for the smallest tour in the graph that visits every vertex exactly once. This version is NPO-complete, in the sense that no approximation algorithm exists unless P = NP [SG76]. If the given edge-weights fulfill the triangle inequality, we obtain the *metric* Tsp ; this variant admits a well-known $\frac{3}{2}$-approximation [Chr76]. We often allow vertices and edges to be used multiple times in a Tsp tour but in this case, their weights are also counted multiple times; furthermore, one can always use the shortest-paths metric in the graph and obtain the metric version. *In this thesis, the* Tsp *usually refers to this version of the problem where repetition of vertices and edges is allowed.* We often refer to the Tsp on planar graphs with shortest-paths metric by *planar* Tsp; similarly, we obtain Tsp on bounded-genus graphs, apex-minor-free graphs, and H-minor-free graphs. The Tsp is known to admit a PTAS on weighted bounded-genus graphs [DHM07] and unweighted apex-minor-free graphs [DHK09]. Whether it admits a PTAS on (weighted) H-minor-free graphs is an important open problem.

The parameterized version of the Hamiltonian Path and Tsp is k-Longest Path, commonly referred to as k-Path: given a graph G and parameter k, we ask whether

G contains a path of length at least k. This problem is in FPT, in fact in EPT, on general graphs [AYZ95] and in SUBEPT on H-minor-free graphs [DFHT05, DH05b]. An important variant is Directed k-Path, the corresponding problem on directed graphs. The problem is in EPT on general directed graphs [AYZ95] and some improvements have been achieved on H-minor-free graphs (see Chapter 6); but whether it is in SUBEPT is an important open question even on planar graphs.

Finally, the most important problem for this thesis is perhaps Steiner Tree. A full introduction to this problem is given in Chapter 1.

A.4 Logic

A signature σ is a finite set of constant symbols $c \in \sigma$ and relation symbols $R \in \sigma$ where each relation symbol is equipped with an *arity* $ar(R)$. A σ-structure $\mathcal{A}$ consists of

(i) a a finite set $U(\mathcal{A})$, called the *universe* of $\mathcal{A}$;
(ii) a constant $c(\mathcal{A}) \in U(\mathcal{A})$ for each constant symbol $c \in \sigma$; and
(iii) for each $R \in \sigma$ an $ar(R)$-ary relation $R(\mathcal{A}) \subseteq (U(\mathcal{A}))^{ar(R)}$.

For signatures σ, τ with $\tau \subseteq \sigma$, we define the *τ-reduct* of a σ-structure $\mathcal{A}$ to be a τ-structure $\mathcal{A}' := \mathcal{A}|_\tau$ with $U(\mathcal{A}') = U(\mathcal{A})$ and having the same constants and relations as $\mathcal{A}$ on all symbols of τ. Conversely, a *σ-expansion* of a τ-structure $\mathcal{A}'$ is a σ-structure $\mathcal{A}$ with $\mathcal{A}|_\tau = \mathcal{A}'$.

We assume to have an unlimited supply of *variables*; these are symbols usually denoted by small letters, such as $x, y, z, v_0, v_1, \ldots$, and one of their purposes is to serve as temporary labels for elements of a structure. A *term* of a signature σ is a constant symbol $c \in \sigma$ or a variable symbol. An *atomic formula* of σ is given by

(i) a string $s = t$, where s and t are σ-terms; or
(ii) $R(t_1, \ldots, t_n)$, where $R \in \sigma$ is an n-ary relation symbol and $t_1, \ldots, t_n$ are σ-terms.

Here and henceforth, we assume that $=$ is not a symbol of the signature σ. Formulas of *first-order logic* over a signature σ are defined inductively as follows:

- every atomic formula of σ is a first-order formula; and
- if φ and ψ are first-order formulas and x is a variable, then $\neg\varphi$, $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$, $\exists x\varphi$, and $\forall x\varphi$ are also first-order formulas, where the symbols $\neg, \vee, \wedge, \exists$, and $\forall$ are intended to mean *not, or, and, exists*, and *for all*, respectively.

We usually denote formulas by Greek letters $\varphi, \psi, \ldots$. We let $\mathrm{FO}[\sigma]$ denote the set of all first-order formulas over σ. We say the quantifier $\forall x$ and $\exists x$ *bind* the variable x. For a formula φ, we call the set of variables that occur in φ but are not bound by a quantifier, the set of *free variables* of φ. A formula without free variables is called a *sentence*. If a formula φ contains free variables $\overline{X} = (x_1, \ldots, x_n)$, we often write $\varphi(\overline{X})$ or $\varphi(x_1, \ldots, x_n)$ to denote this fact; if $T = (t_1, \ldots, t_n)$ are σ-terms, we write $\varphi(T)$ or $\varphi(t_1, \ldots, t_n)$ for the formula obtained from φ by replacing x_i with t_i, $1 \leq i \leq n$. We use the following abbreviations and conventions:

- for terms s, t, we write $s \neq t$ for $\neg(s = t)$;
- we write $\top$ for the formula $\forall x x = x$ and $\bot$ for the formula $\forall x x \neq x$;
- for formulas φ, ψ, we write $(\varphi \to \psi)$ for $(\neg\varphi \vee \psi)$;
- for formulas φ, ψ, we write $(\varphi \leftrightarrow \psi)$ for $(\varphi \to \psi) \wedge (\psi \to \varphi)$;
- $\neg$ is evaluated first, $\vee$ and $\wedge$ are evaluated next, followed by the quantifiers $\exists, \forall$, and $\to, \leftrightarrow$ are evaluated last; we often omit the outermost parentheses of a formula;
- $\bigvee_{i=1}^{k} \varphi(x_i)$ denotes $\varphi(x_i) \vee \cdots \vee \varphi(x_k)$; we freely use such variations with $\bigvee$ and $\bigwedge$;
- $\exists^{\leq k} x \varphi(x)$ denotes $\exists x_1 \ldots \exists x_k \neg\exists y(\varphi(y) \wedge \bigwedge_{i=1}^{k} y \neq x_i)$ intending to mean that there exist at most k elements in the structure fulfilling $\varphi(x)$;
- $\exists^{\geq k} x \varphi(x)$ to mean that there exist at least k elements in the structure fulfilling $\varphi(x)$, i.e. $\neg\exists^{\leq k-1} x\, \varphi(x)$
- $\exists^{=k} x \varphi(x)$ to mean that there exist exactly k elements in the structure fulfilling $\varphi(x)$, i.e. $\exists^{\leq k} x\, \varphi(x) \wedge \exists^{\geq k} x\, \varphi(x)$.

A.4.1 Semantics

A *σ-interpretation* of a formula φ is a pair $\mathcal{I} = (\mathcal{A}, \beta)$, where $\mathcal{A}$ is a σ-structure and $\beta : D \to \mathcal{A}$ is a function, with some domain D that includes the set of free variables of φ, assigning an element of $\mathcal{A}$ to every free variable of φ.

A σ-interpretation $\mathcal{I} = (\mathcal{A}, \beta)$ *satisfies* a formula σ if φ is *true* in the structure $\mathcal{A}$ when the free variables of φ are interpreted by the elements specified by β; here, we evaluate the truth of formulas as follows:

- if φ is an atomic formula of the form $s = t$, then φ is true under $\mathcal{I}$ if and only if the terms s and t are interpreted by the same element of $U(\mathcal{A})$ under $\mathcal{I}$;
- if φ is an atomic formula of the form $R(t_1, \ldots, t_n)$, then φ is true under $\mathcal{I}$ if and only if the terms $t_1, \ldots, t_n$ are interpreted by elements $a_1, \ldots, a_n \in U(\mathcal{A})$ such that $(a_1, \ldots, a_n) \in R(\mathcal{A})$;
- otherwise, we proceed inductively using the standard semantics of the symbols $\neg, \vee, \wedge, \exists$, and $\forall$. For example, the formula $\varphi \wedge \psi$ is true if and only if φ and ψ are true; and the formula $\exists x \varphi(x)$ is true if and only if there exists an element $a \in U(\mathcal{A})$ such that φ becomes true if the free variable x is interpreted by a.

If a σ-interpretation $\mathcal{I}$ satisfies a formula φ we often say that $\mathcal{I}$ *is a model* for φ or $\mathcal{I}$ *models* φ and write $\mathcal{I} \models \varphi$. If φ is a sentence, we also write $\mathcal{A} \models \varphi$ when φ is true in the σ-structure $\mathcal{A}$.

We refer to the book of Ebbinghaus, Flum, and Thomas [EFT94] for more background on mathematical logic.

A.4.2 Monadic Second-Order Logic

The class of formulas of *monadic second-order logic* (MSO) is obtained as the extension of first-order logic by quantification over sets of elements. That is, in addition to first-order variables, which we denote by small letters $x, y, \ldots$, there are variables $X, Y, \ldots$ ranging over sets of elements. Formulas of MSO$[\sigma]$ are then built up inductively by the rules for

first-order logic FO[σ] with the following additional rules: if x is a first-order variable, X a second-order variable, and φ an MSO[σ] formula with free variable X, then (i) Xx is an atomic formula intending to mean that x is an element of X; equivalently, we often write $x \in X$ with the same semantics; (ii) $\exists X\varphi$; and (iii) $\forall X\varphi$ are also MSO[σ] formulas, the semantics being again the obvious standard semantics where, e.g. a formula $\exists X\varphi$ is true in a σ-structure $\mathcal{A}$ if there is a subset $A \subseteq U(\mathcal{A})$ such that φ is true in $\mathcal{A}$ if the variable X is interpreted by A. We write $\mathcal{A} \models \varphi$ to indicate that φ is true in $\mathcal{A}$. We refer to the book of Libkin [Lib04] for more on MSO.

A.4.3 MSO on Graphs

One way to work with graphs in logic is to consider the signature $\sigma = \{E\}$, where E is a binary relation symbol, and define a graph G as a σ-structure $\mathcal{G}$ with universe $U(\mathcal{G}) := V(G)$ and $E(\mathcal{G}) := E(G)$. If we define graphs in this way, then MSO[σ] will only allow us to quantify over vertex sets. This variant of monadic second-order logic over graphs is often referred to as MSO_1 and is indeed important in its own right [Cou90]; however, as in this work we would like to be able to also quantify over edge sets, we take a second standard approach to define graphs as below.

The signature σ_{graph} of *incidence structures* is defined as $\sigma_{graph} := \{V, E, \in\}$, where V, E are unary and $\in$ is a binary relation symbol. We will always use $\in$ in infix notation and write $v \in^A e$ instead of $(v, e) \in \in(A)$. With any graph G we associate a σ_{graph}-structure $\mathcal{G}$, its *incidence structure*, with universe $U(\mathcal{G}) := V(G) \dot\cup E(G)$ and $V(\mathcal{G}) := V(G)$, $E(\mathcal{G}) := E(G)$, and $v \in^{\mathcal{G}} e$ if and only if $v \in V(G), e \in E(G)$, and v and e are incident in G. We will not usually distinguish between a graph G and its incidence structure and write $v \in e$ instead of $v \in^{\mathcal{G}} e$ whenever no confusion is possible. Furthermore, we use the set-theoretic notions of $\in, \subseteq, \cap, \cup, \varnothing$ with their standard obvious semantics. For example, for sets of elements X, Y (which can be V, E, or some second-order variables), we write $X \subseteq Y$ for the formula $\forall x(x \in X \to x \in Y)$ and use the shortcuts $X \neq \varnothing$ and $\forall X \subseteq Y\varphi$ for $\exists x(x \in X)$ and $\forall X(X \subseteq Y \to \varphi)$, respectively.

If we consider MSO[σ_{graph}] we obtain MSO over graphs with quantification over vertex and edge sets, usually denoted by MSO_2. To give an example, the following MSO_2-formula is true in a graph G if and only if G is 3-colorable:

$$\exists C_1 \exists C_2 \exists C_3 \forall x \textstyle\bigvee_{i=1}^{3} x \in C_i \ \wedge \forall e \in E$$
$$(\forall x \in e \forall y \in e(x \neq y \to \textstyle\bigwedge_{1 \leq i \leq 3} \neg(x \in C_i \wedge y \in C_i)))$$

Whereas this property is also definable in MSO_1, we give a further example that is not expressible in MSO_1, namely a formula φ_{Ham} that is true in a graph G if and only if G contains a Hamiltonian path. We derive this formula step-by-step by defining a number of formulas that will also be useful to us later on:

- for a second-order variable P denoting a set of edges, we write $x \in V(P)$ for the formula $\exists e(e \in P \wedge v \in e)$;

- we can define that X consists of connected components of P by

$$\mathit{components}(X,P) := X \subseteq P \subseteq E \wedge \forall e \in P(e \cap V(X) \neq \varnothing \to e \in X)\,;$$

- $\mathit{conn}(P) := \forall X \neq \varnothing(\mathit{components}(X,P) \to X = P)$ states that P is connected;
- $\mathit{deg}^{\leq k}(v,P) := \exists^{\leq k} e \in P(v \in e)$ states that the degree of a vertex v is at most k in P; similarly, $\mathit{deg}^{\geq k}(v,P)$ and $\mathit{deg}^{=k}(v,P)$ can be defined to state that the degree of a vertex v is at least or exactly k in P, respectively;
- $\mathit{ac}(P) := \forall X \subseteq P\, \forall e \in X\, \forall u,v \in e\big(\mathit{conn}(X) \wedge u \neq v \wedge \mathit{deg}^{\geq 2}(u,X) \wedge \mathit{deg}^{\geq 2}(v,X) \to \forall Y(Y = X - e \to \neg\mathit{conn}(Y))\big)$ states that P is acyclic; and
- we can define that P is path by

$$\mathit{path}(P) := \mathit{conn}(P) \wedge \mathit{ac}(P) \wedge \forall v\ \mathit{deg}^{\leq 2}(v,P)\,.$$

Finally, $\varphi_{Ham} := \exists P\, \mathit{path}(P) \wedge V \subseteq V(P)$ expresses that the graph contains a Hamiltonian path, as desired.

A.4.4 The Model-Checking Problem of MSO

The *model-checking problem* MC(MSO) for MSO is defined as the problem, given a structure G and a formula $\varphi \in \mathrm{MSO}$, to decide if $G \models \varphi$. As we saw in the examples above, MSO has a high expressive power, and thus its model-checking problem is naturally of high significance.

In [Var82], Vardi proved that MC(MSO) is PSPACE-complete. However the hardness result crucially uses the fact that the formula is part of the input (and in fact holds on a fixed two-element structure), whereas we are primarily interested in the complexity of checking a fixed formula expressing a graph property in a given input graph. We therefore study model-checking problems in the framework of parameterized complexity:

Let $\mathcal{C}$ be a class of σ-structures. The *parameterized model-checking problem* for MSO on $\mathcal{C}$, denoted by $\mathrm{MC}(\mathrm{MSO},\mathcal{C})$, is defined as the problem to decide, given $G \in \mathcal{C}$ and $\varphi \in \mathrm{MSO}[\sigma]$, if $G \models \varphi$. The *parameter* is $k := |\varphi|$.

As, for instance, the NP-complete problem 3-colorability is definable in MSO_1, we obtain that $\mathrm{MC}(\mathrm{MSO}_1, \textsc{Graphs})$, the model-checking problem for MSO_1 on the class of all graphs, is not fixed-parameter tractable unless P = NP. However, Courcelle's seminal theorem [Cou90] states that if we restrict the class of admissible input graphs, then we can obtain much better results: he showed that $\mathrm{MC}(\mathrm{MSO}_2,\mathcal{C})$ is fixed-parameter tractable with parameter $|\varphi|$ on any class $\mathcal{C}$ of graphs of treewidth bounded by a constant.

In the last part of this thesis, we will prove (under suitable complexity assumptions) the first *lower bounds* for this theorem; namely, that there exist natural classes of graphs of small (polylogarithmic) but unbounded treewidth that do not admit a fixed-parameter tractable model-checking algorithm for MSO_2.

Bibliography

[ACC+96] S. R. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. H. M. Smid, C. D. Zaroliagis: Planar spanners and approximate shortest path queries among obstacles in the plane. In: *ESA '96: Proceedings of the 4th annual European Symposium on Algorithms*, pp. 514–528. Springer, 1996.

[ACG+03] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi: *Complexity and Approximation.* Springer, 2nd edition, 2003.

[ADN05] J. Alber, F. Dorn, R. Niedermeier: Experimental evaluation of a tree decomposition-based algorithm for vertex cover on planar graphs. In: *Discrete Applied Mathematics*, vol. 145(2):pp. 219–231, 2005.

[AFN04] J. Alber, H. Fernau, R. Niedermeier: Parameterized complexity: exponential speed-up for planar graph problems. In: *Journal of Algorithms*, vol. 52(1):pp. 26–56, 2004.

[AGK+98] S. Arora, M. Grigni, D. Karger, P. Klein, A. Woloszyn: A polynomial-time approximation scheme for weighted planar graph TSP. In: *SODA '98: Proceedings of the 9th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 33–41. 1998.

[AKK99] S. Arora, D. R. Karger, M. Karpinski: Polynomial time approximation schemes for dense instances of NP-hard problems. In: *J. Comput. Syst. Sci.*, vol. 58(1):pp. 193–210, 1999.

[AP89] S. Arnborg, A. Proskurowski: Linear time algorithms for NP-hard problems restricted to partial k-trees. In: *Discrete Applied Mathematics*, vol. 23(1):pp. 11–24, 1989.

[Aro98] S. Arora: Polynomial time approximation schemes for the Euclidean traveling salesman and other geometric problems. In: *Journal of the ACM*, vol. 45:pp. 753–782, 1998.

[Aro03] S. Arora: Approximation schemes for NP-hard geometric optimization problems: a survey. In: *Mathematical Programming*, vol. 97(1–2):pp. 43–69, 2003.

[AST90] N. Alon, P. Seymour, R. Thomas: A separator theorem for nonplanar graphs. In: *Journal of the AMS*, vol. 3(4):pp. 801–808, 1990.

[AYZ95] N. Alon, R. Yuster, U. Zwick: Color-coding. In: *J. ACM*, vol. 42(4):pp. 844–856, 1995.

[Bak94] B. S. Baker: Approximation algorithms for NP-complete problems on planar graphs. In: *J. ACM*, vol. 41(1):pp. 153–180, 1994.

[BB91] M. Bern, D. Bienstock: Polynomially solvable special cases of the Steiner problem in planar networks. In: *Annals of Operation Research*, vol. 33:pp. 403–418, 1991.

[BBR07] E. Birmelé, J. A. Bondy, B. A. Reed: Brambles, prisms and grids. In: *Graph theory in Paris*, Trends Math., pp. 37–44. Birkhäuser, 2007.

[BDT09] G. Borradaile, E. D. Demaine, S. Tazari: Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. In: *STACS '09: Proceedings of the 26th Symposium on Theoretical Aspects of Computer Science*, pp. 171–182. 2009.

[Ber90] M. Bern: Faster exact algorithms for Steiner trees in planar networks. In: *Networks*, vol. 20:pp. 109–120, 1990.

[BGHK95] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, T. Kloks: Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. In: *J. Algorithms*, vol. 18(2):pp. 238–255, 1995.

[BGK05] H. L. Bodlaender, A. Grigoriev, A. M. C. A. Koster: Treewidth lower bounds with brambles. In: *ESA '05: Proceedings of the 13th annual European Symposium on Algorithms*, vol. 3669 of *LNCS*, pp. 391–402. Springer, 2005.

[BGRS10] J. Byrka, F. Grandoni, T. Rothvoß, L. Sanità: An improved LP-based approximation for Steiner tree. In: *STOC '10: Proceedings of the 42nd annual ACM Symposium on Theory of computing*, p. to appear. ACM, 2010.

[BH92] R. Boppana, M. M. Halldórsson: Approximating maximum independent sets by excluding subgraphs. In: *Bit*, vol. 32:pp. 180–196, 1992.

[BHKK07] A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto: Fourier meets Möbius: fast subset convolution. In: *STOC '07: Proceedings of the 39th annual ACM Symposium on Theory of Computing*, pp. 67–74. ACM, 2007.

[BHM10] M. Bateni, M. Hajiaghayi, D. Marx: Approximation schemes for steiner forest on planar graphs and graphs of bounded treewidth. In: *STOC '10: Proceedings of the 42nd annual ACM Symposium on Theory of Computing*, p. to appear. ACM, 2010.

[BK08a] H. L. Bodlaender, A. M. C. A. Koster: Combinatorial optimization on graphs of bounded treewidth. In: *The Computer Journal*, vol. 51(3):pp. 255–269, 2008.

[BK08b] G. Borradaile, P. Klein: The two-edge connectivity survivable network problem in planar graphs. In: *ICALP '08: Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, vol. 5125 of *LNCS*, pp. 485–501. Springer, 2008.

[BKM07a] G. Borradaile, P. N. Klein, C. Mathieu: A polynomial-time approximation scheme for Steiner tree in planar graphs. In: *SODA '07: Proceedings of the 18th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1285–1294. 2007.

[BKM07b] G. Borradaile, P. N. Klein, C. Mathieu: Steiner tree in planar graphs: An $O(n \log n)$ approximation scheme with singly exponential dependence on epsilon. In: *WADS '07: Proceedings of the 10th Workshop on Algorithms and Data Structures*, vol. 4619 of *LNCS*, pp. 275–286. Springer, 2007.

[BKM09] G. Borradaile, P. N. Klein, C. Mathieu: An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. In: *ACM Transactions on Algorithms*, vol. 5(3), 2009.

[Bod88] H. L. Bodlaender: Dynamic programming on graphs with bounded treewidth. In: *ICALP '88: Proceedings of the 15th International Colloquium on Automata, Languages and Programming*, pp. 105–118. Springer, 1988.

[Bod96] H. L. Bodlaender: A linear-time algorithm for finding tree-decompositions of small treewidth. In: *SIAM J. Comput.*, vol. 25(6):pp. 1305–1317, 1996.

[BP89] M. Bern, P. Plassmann: The Steiner problem with edge lengths 1 and 2. In: *Information Processing Letters*, vol. 32(4):pp. 171–176, 1989.

[BT98] B. Bollobás, A. Thomason: Proof of a conjecture of Mader, Erdös and Hajnal on topological complete subgraphs. In: *Europ. J. Combinatorics*, vol. 19:pp. 883–887, 1998.

[BTW00] M. Brazil, D. Thomas, P. Winter: Minimum networks in uniform orientation metrics. In: *SIAM Journal on Computing*, vol. 30(5):pp. 1579–1593, 2000.

[CC02] M. Chlebík, J. Chlebíková: Approximation hardness of the Steiner tree problem on graphs. In: *SWAT '02: Proceedings of the 8th Scandinavian Workshop on Algorithm Theory*, vol. 2368 of *LNCS*, pp. 170–179. Springer, 2002.

[CC07] S. Cabello, E. W. Chambers: Multiple source shortest paths in a genus g graph. In: *SODA '07: Proceedings of the 18th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 89–97. SIAM, 2007.

[CCK+03] H. Chen, C.-K. Cheng, A. B. Kahng, I. Măndoiu, Q. Wang: Estimation of wirelength reduction for λ-geometry vs. Manhattan placement and routing. In: *SLIP '03: Proceedings of the 5th International Workshop on System-Level Interconnect Prediction*, pp. 71–76. ACM Press, 2003.

[CD01] X. Cheng, D. Du: *Steiner Trees in Industry*. Kluwer Academic Publishers, 2001.

[Cho78] E.-A. Choukhmane: Une heuristique pour le problème de l'arbre de Steiner. In: *Rech. Opèr.*, vol. 12:pp. 207–212, 1978.

[Chr76] N. Christofides: Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Carnegie-Mellon University, Pittsburgh, PA, USA, 1976.

[CKM+98] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov, A. Zelikovsky: On wirelength estimations for row-based placement. In: *ISPD '98: Proceedings of the 1998 International Symposium on Physical Design*, pp. 4–11. ACM, 1998.

[Cla87] K. Clarkson: Approximation algorithms for shortest path motion planning. In: *STOC '87: Proceedings of the 19th annual ACM Symposium on Theory of Computing*, pp. 56–65. ACM Press, 1987.

[CMT09] M. Chapelle, F. Mazoit, I. Todinca: Constructing brambles. In: *MFCS 09': Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science*. 2009. To appear.

[Coo71] S. A. Cook: The complexity of theorem-proving procedures. In: *STOC '71: Proceedings of the 3rd annual ACM Symposium on Theory of Computing*, pp. 151–158. ACM, 1971.

[Cou90] B. Courcelle: Graph rewriting: An algebraic and logic approach. In: *Handbook of Theoretical Computer Science*, vol. 2, pp. 194–242. Elsevier, 1990.

[CSH08] V. Chandrasekaran, N. Srebro, P. Harsha: Complexity of inference in graphical models. In: *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*, pp. 70–78. AUAI Press, 2008.

[CT76] D. R. Cheriton, R. E. Tarjan: Finding minimum spanning trees. In: *SIAM Journal on Computing*, vol. 5(4):pp. 724–742, 1976.

[DF95a] R. G. Downey, M. R. Fellows: Fixed-parameter tractability and completeness I: Basic results. In: *SIAM J. Comput.*, vol. 24:pp. 873–921, 1995.

[DF95b] R. G. Downey, M. R. Fellows: Fixed-parameter tractability and completeness II: On completeness for W[1]. In: *Theoretical Computer Science*, vol. 141:pp. 109–131, 1995.

[DF95c] R. G. Downey, M. R. Fellows: Parameterized computational feasibility. In: *Proceedings of Feasible Mathematics II*, pp. 219–244. Birkhäuser, 1995.

[DF99] R. G. Downey, M. R. Fellows: *Parameterized Complexity*. Springer, 1999.

[DFHT05] E. D. Demaine, F. V. Fomin, M. Hajiaghayi, D. M. Thilikos: Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. In: *J. ACM*, vol. 52(6):pp. 866–893, 2005.

[DFL+10] F. Dorn, F. V. Fomin, D. Lokshtanov, V. Raman, S. Saurabh: Beyond bidimensionality: Parameterized subexponential algorithms on directed graphs. In: *STACS '10: Proceedings of the 27th Symposium on Theoretical Aspects of Computer Science*, pp. 251–262. 2010.

[DFT08] F. Dorn, F. V. Fomin, D. M. Thilikos: Catalan structures and dynamic programming in H-minor-free graphs. In: *SODA '08: Proceedings of the 19th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 631–640. SIAM, 2008.

[DGJT99] R. Diestel, K. Gorbunov, T. Jensen, C. Thomassen: Highly connected sets and the excluded grid theorem. In: *J. Combin. Theory (Series B)*, vol. 75:pp. 61–73, 1999.

[DGK07] A. Dawar, M. Grohe, S. Kreutzer: Locally excluding a minor. In: *LICS '07: Proceedings of the 22nd annual IEEE symposium on Logic in Computer Science*, pp. 270–279. IEEE Computer Society, 2007.

[DH04] E. D. Demaine, M. Hajiaghayi: Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In: *SODA '04: Proceedings of the 15th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 840–849. SIAM, 2004.

[DH05a] E. D. Demaine, M. Hajiaghayi: Bidimensionality: new connections between FPT algorithms and PTASs. In: *SODA '05: Proceedings of the 16th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 590–601. 2005.

[DH05b] E. D. Demaine, M. Hajiaghayi: Graphs excluding a fixed minor have grids as large as treewidth, with combinatorial and algorithmic applications through bidimensionality. In: *SODA '05: Proceedings of the 16th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 682–689. SIAM, 2005.

[DH07] E. D. Demaine, M. Hajiaghayi: Quickly deciding minor-closed parameters in general graphs. In: *Eur. J. Comb.*, vol. 28(1):pp. 311–314, 2007.

[DHK05] E. D. Demaine, M. Hajiaghayi, K. Kawarabayashi: Algorithmic graph minor theory: Decomposition, approximation, and coloring. In: *FOCS '05: Proceedings of the 46th annual IEEE Symposium on Foundations of Computer Science*, pp. 637–646. IEEE Computer Society, 2005.

[DHK09] E. D. Demaine, M. Hajiaghayi, K. Kawarabayashi: Approximation algorithms via structural results for apex-minor-free graphs. In: *ICALP '09: Proceedings of the 36th International Colloquium on Automata, Languages and Programming, Part I*, vol. 5555 of *LNCS*, pp. 316–327. Springer, 2009.

[DHK10] E. D. Demaine, M. Hajiaghayi, K. Kawarabayashi: Decomposition, approximation, and coloring of odd-minor-free graphs. In: *SODA '10: Proceedings of the 21st annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 329–344. SIAM, 2010.

[DHM07] E. D. Demaine, M. Hajiaghayi, B. Mohar: Approximation algorithms via contraction decomposition. In: *SODA '07: Proceedings of the 18th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 278–287. SIAM, 2007.

[Die05] R. Diestel: *Graph Theory*. Springer, 2005.

[Dij59] E. Dijkstra: A note on two problems in connexion with graphs. In: *Numerische Mathematik 1*, pp. 269–271, 1959.

[Dor10] F. Dorn: Planar subgraph isomorphism revisited. In: *STACS '10: Proceedings of the 27th Symposium on Theoretical Aspects of Computer Science*, pp. 263–274. 2010.

[DW72] S. E. Dreyfus, R. A. Wagner: The Steiner problem in graphs. In: *Networks*, vol. 1:pp. 195–208, 1972.

[EFT94] H.-D. Ebbinghaus, J. Flum, W. Thomas: *Mathematical Logic*. Springer, 2nd edition, 1994.

[EL75] P. Erdös, L. Lovász: Problems and results on 3-chormatic hypergraphs and some related questions. In: *Infinite and Finite Sets*, vol. 10 of *Colloq. Math. Soc. János Bolyai*, pp. 609–627. North Holland, 1975.

[EMAFV87] R. E. Erickson, C. L. Monma, J. Arthur F. Veinott: Send-and-split method for minimum-concave-cost network flows. In: *Math. Oper. Res.*, vol. 12(4):pp. 634–664, 1987.

[Epp00a] D. Eppstein: Diameter and treewidth in minor-closed graph families. In: *Algorithmica*, vol. 27(3):pp. 275–291, 2000.

[Epp00b] D. Eppstein: Spanning trees and spanners. In: *Handbook of Computational Geometry*, chapter 9, pp. 425–461. Elsevier, 2000.

[Epp03] D. Eppstein: Dynamic generators of topologically embedded graphs. In: *SODA '03: Proceedings of the 14th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 599–608. SIAM, 2003.

[EW05] J. Erickson, K. Whittlesey: Greedy optimal homotopy and homology generators. In: *SODA '05: Proceedings of the 16th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1038–1046. SIAM, 2005.

[FG82] L. Foulds, R. Graham: The Steiner problem in phlogeny is NP-complete. In: *Advances in Applied Mathematics*, vol. 3:pp. 43–49, 1982.

[FG01] M. Frick, M. Grohe: Deciding first-order properties of locally tree-decomposable structures. In: *J. ACM*, vol. 48(6):pp. 1184–1206, 2001.

[FG06] J. Flum, M. Grohe: *Parameterized Complexity Theory*. Springer, 2006.

[FHL08] U. Feige, M. Hajiaghayi, J. R. Lee: Improved approximation algorithms for minimum weight vertex separators. In: *SIAM J. Comput.*, vol. 38(2):pp. 629–657, 2008.

[FL88] M. R. Fellows, M. A. Langston: Nonconstructive tools for proving polynomial-time decidability. In: *J. ACM*, vol. 35(3):pp. 727–739, 1988.

[Fre85] G. N. Frederickson: Data structures for on-line updating of minimum spanning trees, with applications. In: *SIAM Journal on Computing*, vol. 14(4):pp. 781–798, November 1985.

[Fre87] G. N. Frederickson: Fast algorithms for shortest paths in planar graphs, with applications. In: *SIAM Journal on Computing*, vol. 16(6):pp. 1004–1022, December 1987.

[FT87] M. L. Fredman, R. E. Tarjan: Fibonacci heaps and their uses in improved network optimization algorithms. In: *J. ACM*, vol. 34(3):pp. 596–615, 1987.

[GC94] J. L. Ganley, J. P. Cohoon: Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles. In: *ISCAS '94: Proceedings of the 1994 International Symposium on Circuits and Systems*, pp. 113–116. 1994.

[GGG+04] J. Geelan, B. Gerards, L. Goddyn, B. Reed, P. Seymour, A. Vetta: The odd case of Hadwiger's conjecture, 2004. Submitted.

[GGJ77] M. Garey, R. Graham, D. Johnson: The complexity of computing Steiner minimal trees. In: *SIAM Journal on Applied Mathematics*, vol. 32:pp. 835–859, 1977.

[GJ77] M. R. Garey, D. S. Johnson: The rectilinear Steiner tree problem is NP-complete. In: *SIAM Journal on Applied Mathematics*, vol. 32:pp. 826–834, 1977.

[GJ79] M. R. Garey, D. S. Johnson: *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, NY, USA, 1979.

[GKP95] M. Grigni, E. Koutsoupias, C. Papadimitriou: An approximation scheme for planar graph TSP. In: *FOCS '95: Proceedings of the 36th annual IEEE Symposium on Foundations of Computer Science*, pp. 640–645. IEEE Computer Society, 1995.

[GM09] M. Grohe, D. Marx: On tree width, bramble size, and expansion. In: *J. Comb. Theory, Ser. B*, vol. 99(1):pp. 218–228, 2009.

[GN07] J. Guo, R. Niedermeier: Invitation to data reduction and problem kernelization. In: *SIGACT News*, vol. 38(1):pp. 31–45, 2007.

[Gol01] A. V. Goldberg: A simple shortest path algorithm with linear average time. In: *ESA '01: Proceedings of the 9th annual European Symposium on Algorithms*, vol. 2161 of *LNCS*, pp. 230–241. Springer, 2001.

[GP68] E. N. Gilbert, H. O. Pollak: Steiner minimal trees. In: *SIAM Journal on Applied Mathematics*, vol. 16(1):pp. 1–29, 1968.

[Gro03] M. Grohe: Local tree-width, excluded minors, and approximation algorithms. In: *Combinatorica*, vol. 23(4):pp. 613–632, 2003.

[Gro04] M. Grohe: Computing crossing numbers in quadratic time. In: *J. Comput. Syst. Sci.*, vol. 68(2):pp. 285–302, 2004.

[Gro07a] M. Grohe: Logic, graphs, and algorithms. In: *Logic and Automata – History and Perspectives*. Amsterdam University Press, 2007.

[Gro07b] M. Grohe: The complexity of homomorphism and constraint satisfaction problems seen from the other side. In: *J. ACM*, vol. 54(1), 2007.

[GRSZ94] J. Griffith, G. Robins, J. S. Salowe, T. Zhang: Closing the gap: Near-optimal Steiner trees in polynomial time. In: *IEEE Trans. Computer-Aided Design*, vol. 13:pp. 1351–1365, 1994.

[Gue01] B. Guenin: A characterization of weakly bipartite graphs. In: *J. Comb. Theory Ser. B*, vol. 83(1):pp. 112–168, 2001.

[GW95] M. X. Goemans, D. P. Williamson: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. In: *J. ACM*, vol. 42:pp. 1115–1145, 1995.

[Had43] H. Hadwiger: Über eine Klassifikation der Streckenkomplexe. In: *Vierteljschr. Naturforsch. Ges. Zürich*, vol. 88:pp. 133–142, 1943.

[Hag00] T. Hagerup: Improved shortest paths on the word RAM. In: *ICALP '00: Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, vol. 1853 of *LNCS*, pp. 61–72. Springer, 2000.

[Hal02] E. Halperin: Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. In: *SIAM J. Comput.*, pp. 1608–1623, 2002.

[Han66] M. Hanan: On Steiner's problem with rectilinear distance. In: *SIAM Journal on Applied Mathematics*, vol. 14:pp. 255–265, 1966.

[HKK05] I. V. Hicks, A. M. C. A. Koster, E. Kolotoğlu: Branch and tree decomposition techniques for discrete optimization. In: *TutORials 2005*, INFORMS TutORials in Operations Research Series, chapter 1, pp. 1–29. INFORMS annual Meeting, 2005.

[HKRS97] M. R. Henzinger, P. N. Klein, S. Rao, S. Subramanian: Faster shortest-path algorithms for planar graphs. In: *Journal of Computer and System Sciences*, vol. 55(1):pp. 3–23, 1997.

[Hod97] W. Hodges: *A shorter model theory.* Cambridge University Press, 1997.

[HRW92] F. Hwang, D. Richards, P. Winter: *The Steiner Tree Problem*, vol. 53. Annals of Discrete Mathematics, North-Holland, 1992.

[Hwa76] F. Hwang: On Steiner minimal trees with rectilinear distance. In: *SIAM Journal on Applied Mathematics*, vol. 30:pp. 104–114, 1976.

[Joh74] D. S. Johnson: Approximation algorithms for combinatorial problems. In: *J. Comput. System Sci*, vol. 9:pp. 256–278, 1974.

[Kar72] R. M. Karp: Reducibility among combinatorial problems. In: *Complexity of Computer Computations*, pp. 85–103. Plenum Press, 1972.

[Kle05] P. N. Klein: Multiple-source shortest paths in planar graphs. In: *SODA '05: Proceedings of the 16th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 146–155. 2005.

[Kle06] P. N. Klein: A subset spanner for planar graphs, with application to subset TSP. In: *STOC '06: Proceedings of the 38th annual ACM Symposium on Theory of Computing*, pp. 749–756. 2006.

[Kle08] P. N. Klein: A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights. In: *SIAM J. Comput.*, vol. 37(6):pp. 1926–1952, 2008.

[KLR10] K. Kawarabayashi, Z. Li, B. Reed: Recognizing a totally odd K_4-subdivision, parity 2-disjoint rooted paths and parity cycle through specified elements. In: *SODA '10: Proceedings of the 21st annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 318–328. SIAM, 2010.

[KM98] T. Koch, A. Martin: Solving Steiner tree problems in graphs to optimality. In: *Networks*, vol. 33:pp. 207–232, 1998.

[KM08] K. Kawarabayashi, B. Mohar: Approximating chromatic number and list-chromatic number of minor-closed and odd-minor-closed classes of graphs. In: *STOC '06: Proceedings of the 38th annual ACM Symposium on Theory of Computing*, pp. 401–406. ACM Press, 2008.

[KMV01] T. Koch, A. Martin, S. Voß: SteinLib: An updated library on Steiner tree problems in graphs. In: *Steiner Trees in Industries*, pp. 285–325. Kluwer, 2001. ZIB-Report 00-37.

[KMZ03] A. Kahng, I. Măndoiu, A. Zelikovsky: Highly scalable algorithms for rectilinear and octilinear Steiner trees. In: *Proceedings 2003 Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 827–833, 2003.

[Kön31] D. König: Gráfok és mátrixok. In: *Matematikai és Fizikai Lapok*, vol. 38:pp. 116–119, 1931.

[KR92] A. Kahng, G. Robins: A new class of iterative Steiner tree heuristics with good performance. In: *IEEE Trans. on CAD*, vol. 11:pp. 1462–1465, 1992.

[KR95] A. Kahng, G. Robins: *On Optimal Interconnections for VLSI*. Kluwer Publishers, 1995.

[Kre09a] S. Kreutzer: Algorithmic meta-theorems. In: *Electronic Colloquium on Computational Complexity (ECCC)*, vol. TR09-147, 2009.

[Kre09b] S. Kreutzer: On the parameterised intractability of monadic second-order logic. In: *CSL '09: Proceedings of the 18th annual conference on Computer Science Logic*. 2009. 348–363.

[KS81] B. Korte, R. Schrader: On the existence of fast approximation schemes. In: *Nonlinear Programming*, vol. 4:pp. 415–437, 1981.

[KS90] E. Korach, N. Solel: Linear time algorithm for minimum weight Steiner tree in graphs with bounded treewidth. Manuscript, 1990.

[KT10a] S. Kreutzer, S. Tazari: Lower bounds for the complexity of monadic second-order logic. In: *LICS '10: Proceedings of the 25th annual IEEE symposium on Logic in Computer Science*, p. to appear. IEEE Computer Society, 2010.

[KT10b] S. Kreutzer, S. Tazari: On brambles, grid-like minors, and parameterized intractability of monadic second-order logic. In: *SODA '10: Proceedings of the 21st annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 354–364. SIAM, 2010.

[Kur30] K. Kuratowski: Sur le problème des courbes gauches en topologie. In: *Fund. Math*, vol. 15:pp. 271–283, 1930.

[KvHK02] A. M. C. A. Koster, C. P. M. van Hoesel, A. W. J. Kolen: Solving partial constraint satisfaction problems with tree decomposition. In: *Networks*, vol. 40(3):pp. 170–180, 2002.

[Lib04] L. Libkin: *Elements of Finite Model Theory.* Springer, 2004.

[Lic82] D. Lichtenstein: Planar formulae and their uses. In: *SIAM Journal on Computing*, vol. 11(2):pp. 329–343, 1982.

[LR88] F. T. Leighton, S. Rao: An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In: *FOCS '88: Proceedings of the 29th annual IEEE Symposium on Foundations of Computer Science*, pp. 422–431. IEEE, 1988.

[LT79] R. J. Lipton, R. E. Tarjan: A separator theorem for planar graphs. In: *SIAM Journal on Applied Mathematics*, vol. 36(2):pp. 177–189, 1979.

[LT80] R. J. Lipton, R. E. Tarjan: Applications of a planar separator theorem. In: *SIAM Journal on Computing*, vol. 9(3):pp. 615–627, 1980.

[LZSK02] J. Liu, Y. Zhao, E. Shragowitz, G. Karypis: A polynomial time approximation scheme for rectilinear Steiner minimum tree construction in the presence of obstacles. In: *ICECS '02: Proceedings of the 9th IEEE International Conference on Electronics, Circuits and Systems*, vol. 2, pp. 781–784. 2002.

[Mad67] W. Mader: Homomorphieeigenschaften und mittlere Kantendichte von Graphen. In: *Math. Ann.*, vol. 174:pp. 265–268, 1967.

[Mad72] W. Mader: Existenz n-fach zusammenhängender Teilgraphen in Graphen genügend grosser Kantendichte. In: *Abh. Math. Sem. Hamburg Univ.*, vol. 37:pp. 86–97, 1972.

[Mar04] M. Mares: Two linear time algorithms for MST on minor closed graph classes. In: *Archivum mathematicum*, vol. 40(3):pp. 315–320, 2004.

[Mar07] D. Marx: On the optimality of planar and geometric approximation schemes. In: *FOCS '07: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pp. 338–348. IEEE, 2007.

[Mar08] D. Marx: Parameterized complexity and approximation algorithms. In: *The Computer Journal*, vol. 51(1):pp. 60–78, 2008.

[Meh88] K. Mehlhorn: A faster approximation algorithm for the Steiner problem in graphs. In: *Information Processing Letters*, vol. 27:pp. 125–128, 1988.

[Men27] K. Menger: Zur allgemeinen Kurventheorie. In: *Fund. Math.*, vol. 10:pp. 96–115, 1927.

[Mit99] J. S. B. Mitchell: Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. In: *SIAM Journal on Computing*, vol. 28(4):pp. 1298–1309, 1999.

[MM03] J. A. Makowsky, J. Mariño: Treewidth and the monadic quantifier hierarchy. In: *Theor. Comput. Sci.*, vol. 1(303):pp. 157–170, 2003.

[Moh99] B. Mohar: A linear time algorithm for embedding graphs in an arbitrary surface. In: *SIAM Journal on Discrete Mathematics*, vol. 12(1):pp. 6–26, 1999.

[Mos09] R. A. Moser: A constructive proof of the Lovász Local Lemma. In: *STOC '09: Proceedings of the 41st annual ACM Symposium on Theory of computing*, pp. 343–350. ACM, 2009.

[MP03] M. Müller-Hannemann, S. Peyer: Approximation of rectilinear Steiner trees with length restrictions on obstacles. In: *8th Workshop on Algorithms and Data Structures (WADS 2003)*, vol. 2748 of *LNCS*, pp. 207–218. Springer, 2003.

[MS06] M. Müller-Hannemann, A. Schulze: Approximation of octilinear Steiner trees constrained by hard and soft obstacles. In: *SWAT '06: Proceedings of the 10th Scandinavian Workshop on Algorithm Theory*, vol. 4059 of *LNCS*, pp. 242–254. Springer, 2006.

[MS07] M. Müller-Hannemann, A. Schulze: Hardness and approximation of octilinear Steiner trees. In: *International Journal on Computational Geometry and Applications (IJCGA)*, vol. 17:pp. 231–260, 2007.

[MT01] B. Mohar, C. Thomassen: *Graphs on Surfaces*. The John Hopkins University Press, 2001.

[MT07] M. Müller-Hannemann, S. Tazari: A near linear time approximation scheme for Steiner tree among obstacles in the plane. In: *WADS '07: Proceedings of the 10th Workshop on Algorithms and Data Structures*, vol. 4619 of *LNCS*, pp. 151–162. Springer, 2007.

[MT09] R. A. Moser, G. Tardos: A constructive proof of the general Lovász Local Lemma, 2009. Available at arXiv:0903.0544v3 [cs.DS].

[MT10] M. Müller-Hannemann, S. Tazari: A near linear time approximation scheme for Steiner tree among obstacles in the plane. In: *Computational Geoemtry: Theory and Applications*, vol. 43:pp. 395–409, May 2010. Special Issue on WADS 2007.

[Nie02] R. Niedermeier: Invitation to fixed-parameter algorithms. Habilitation thesis, Universität Tübingen, Germany, 2002.

[NWZ02] B. Nielsen, P. Winter, M. Zachariasen: An exact algorithm for the uniformly-oriented Steiner tree problem. In: *10th annual European Symposium on Algorithms (ESA 2002)*, vol. 2461 of *LNCS*, pp. 760–772. Springer, 2002.

[OGD07] Open Graph Drawing Framework, 2007. www.ogdf.net, University of Dortmund, Germany.

[PD01] T. Polzin, S. V. Daneshmand: Improved algorithms for the Steiner problem in networks. In: *Discrete Appl. Math.*, vol. 112(1-3):pp. 263–300, 2001.

[PD02] T. Polzin, S. V. Daneshmand: Extending reduction techniques for the Steiner tree problem. In: *ESA '02: Proceedings of the 10th annual European Symposium on Algorithms*, pp. 795–807. Springer, 2002.

[PD06] T. Polzin, S. V. Daneshmand: Practical partitioning-based methods for the Steiner problem. In: *WEA '06: Proceedings of the 5th International Workshop on Experimental Algorithms*, vol. 4007 of *LNCS*, pp. 241–252. 2006.

[Ple81] J. Plesnik: A bound for the Steiner problem in graphs. In: *Math. Slovaca*, vol. 31:pp. 155–163, 1981.

[PR05] S. Pettie, V. Ramachandran: A shortest path algorithm for real-weighted undirected graphs. In: *SIAM J. Comput.*, vol. 34(6):pp. 1398–1431, 2005.

[Pro88] J. S. Provan: An approximation scheme for finding Steiner trees with obstacles. In: *SIAM Journal on Computing*, vol. 17(5):pp. 920–934, 1988.

[PS02] H. Prömel, A. Steger: *The Steiner Tree Problem: A Tour through Graphs, Algorithms, and Complexity.* Advanced Lectures in Mathematics, Vieweg, 2002.

[PW02] M. Poggi de Aragão, R. F. Werneck: On the implementation of MST-based heuristics for the Steiner problem in graphs. In: *ALENEX '02: Revised Papers from the 4th International Workshop on Algorithm Engineering and Experiments*, pp. 1–15. Springer, 2002.

[PWZ04] M. Paluszewski, P. Winter, M. Zachariasen: A new paradigm for general architecture routing. In: *GLVLSI '04: Proceedings of the 14th ACM Great Lakes Symposium on VLSI*, pp. 202–207. 2004.

[PY91] C. H. Papadimitriou, M. Yannakakis: Optimization, approximation, and complexity classes. In: *J. Comput. System Sci.*, vol. 43:pp. 425–440, 1991.

[Ree92] B. Reed: Finding approximate separators and computing tree width quickly. In: *STOC '92: Proceedings of the 24th annual ACM Symposium on Theory of Computing*, pp. 221–228. ACM, 1992.

[Ree97] B. Reed: Tree Width and Tangles: A New Connectivity Measure And Some Applications. In: *Survey in Combinatorics*, vol. 241:pp. 87–158, 1997.

[RS86] N. Robertson, P. D. Seymour: Graph minors. II. Algorithmic aspects of tree-width. In: *Journal of Algorithms*, vol. 7(3):pp. 309–322, 1986.

[RS95] N. Robertson, P. D. Seymour: Graph minors. XIII. The disjoint paths problem. In: *J. Comb. Theory Ser. B*, vol. 63(1):pp. 65–110, 1995.

[RS98] S. B. Rao, W. D. Smith: Approximating geometrical graphs via "spanners" and "banyans". In: *STOC '98: Proceedings of the 30th annual ACM Symposium on Theory of Computing*, pp. 540–550. ACM Press, 1998.

[RS03] N. Robertson, P. Seymour: Graph minors. XVI. Excluding a non-planar graph. In: *J. Comb. Theory Ser. B*, vol. 89(1):pp. 43–76, 2003.

[RS04] N. Robertson, P. D. Seymour: Graph minors. XX. Wagner's conjecture. In: *J. Comb. Theory Ser. B*, vol. 92(2):pp. 325–357, 2004.

[RS07] V. Raman, S. Saurabh: Improved fixed parameter tractable algorithms for two "edge" problems: MAXCUT and MAXDAG. In: *Information Processing Letters*, vol. 104(2):pp. 65–72, 2007.

[RST94] N. Robertson, P. Seymour, R. Thomas: Quickly excluding a planar graph. In: *J. Comb. Theory Ser. B*, vol. 62(2):pp. 323–348, 1994.

[RW08] B. Reed, D. R. Wood: Polynomial treewidth forces a large grid-like minor, 2008. Available at arXiv:0809.0724v3 [math.CO].

[RW09] B. Reed, D. R. Wood: A linear-time algorithm to find a separator in a graph excluding a minor. In: *ACM Transactions on Algorithms*, vol. 5(4):pp. 1–16, 2009.

[RZ00] G. Robins, A. Zelikovsky: Improved Steiner tree approximation in graphs. In: *SODA '00: Proceedings of the 11th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 770–779. 2000.

[SG76] S. K. Sahni, T. F. Gonzales: P-complete approximation problems. In: *J. ACM*, vol. 23:pp. 555–565, 1976.

[SH76] M. I. Shamos, D. Hoey: Geometric intersection problems. In: *FOCS '76: Proceedings of the 17th annual IEEE Symposium on Foundations of Computer Science*, pp. 208–215. IEEE Computer Society, 1976.

[ST93] P. D. Seymour, R. Thomas: Graph searching and a min-max theorem for tree-width. In: *J. Comb. Theory, Ser. B*, vol. 58(1):pp. 22–33, 1993.

[Tam03] H. Tamaki: A linear time heuristic for the branch-decomposition of planar graphs. In: *ESA '03: Proceedings of the 11th European Symposium on Algorithms*, vol. 2832 of *LNCS*, pp. 765–775. Springer, 2003.

[Taz06] S. Tazari: Algorithmic approaches for two fundamental optimization problems: Workload-balancing and planar Steiner trees. Diploma thesis, Technische Universität Darmstadt, Germany, August 2006.

[Tei02] S. L. Teig: The X architecture: not your father's diagonal wiring. In: *SLIP '02: Proceedings of the 4th International Workshop on System-Level Interconnect Prediction*, pp. 33–37. ACM Press, 2002.

[Tho99] M. Thorup: Undirected single-source shortest paths with positive integer weights in linear time. In: *J. ACM*, vol. 46(3):pp. 362–394, 1999.

[Tho01] A. Thomason: The extremal function for complete minors. In: *J. Comb. Theory Ser. B*, vol. 81(2):pp. 318–338, 2001.

[TM08] S. Tazari, M. Müller-Hannemann: A faster shortest-paths algorithm for minor-closed graph classes. In: *WG '08: Proceedings of the 34th Workshop on Graph Theoretic Concepts in Computer Science*, vol. 5344 of *LNCS*, pp. 360–371. Springer, 2008.

[TM09a] S. Tazari, M. Müller-Hannemann: Dealing with large hidden constants: Engineering a planar steiner tree PTAS. In: *ALENEX '09: Proceedings of the 10th International Workshop on Algorithm Engineering and Experiments*, pp. 120–131. SIAM, 2009.

[TM09b] S. Tazari, M. Müller-Hannemann: Shortest paths in linear time on minor-closed graph classes, with an application to Steiner tree approximation. In: *Discrete Applied Mathematics*, vol. 157:pp. 673–684, 2009.

[TW05] R. E. Tarjan, R. F. Werneck: Self-adjusting top trees. In: *SODA '05: Proceedings of the 16th annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 813–822. SIAM, 2005.

[Var82] M. Vardi: On the complexity of relational query languages. In: *STOC '82: Proceedings of the 14th annual ACM Symposium on Theory of Computing*, pp. 137–146. 1982.

[War97] D. M. Warme: A new exact algorithm for rectilinear Steiner trees. In: *Network Design: Connectivity and Facilities Location: DIMACS Workshop April 28-30, 1997*, pp. 357–395. AMS, 1997.

[WWZ00] D. M. Warme, P. Winter, M. Zachariasen: Exact algorithms for plane Steiner tree problems: A computational study. In: *Advances in Steiner Trees*, pp. 81–116. Kluwer, 2000.

[WWZ03] D. M. Warme, P. Winter, M. Zachariasen: GeoSteiner 3.1. DIKU, Department of Computer Science, Copenhagen, Denmark, http://www.diku.dk/geosteiner/, 2003.

[yEd10] yEd Graph Editor, 2010. www.yworks.com/en/products_yed_about.html, yWorks GmbH, Tübingen, Germany.

[Zeh02] N. Zeh: *I/O-Efficient Algorithms for Shortest Path Related Problems.* Ph.D. thesis, Carleton University, Ottawa, Canada, 2002.

[Zha05] H. Zhao: *Algorithms and Complexity Analyses for Some Combinatorial Optimization Problems.* Ph.D. thesis, New Jersey Institute of Technology, NJ, USA, 2005.

[ZW99] M. Zachariasen, P. Winter: Obstacle-avoiding Euclidean Steiner trees in the plane: An exact approach. In: *ALENEX '99: Selected Papers from the 1st Workshop on Algorithm Engineering and Experimentation*, vol. 1619 of *LNCS*, pp. 282–295. Springer, 1999.

[Zwi01] U. Zwick: Exact and approximate distances in graphs - a survey. In: *ESA '01: Proceedings of the 9th annual European Symposium on Algorithms*, vol. 2161 of *LNCS*, pp. 33–48. Springer, 2001.

Erklärung

Ich erkläre hiermit, dass

- ich die vorliegende Dissertationsschrift mit dem Titel “Algorithmic Graph Minor Theory: Approximation, Parameterized Complexity, and Practical Aspects” selbständig und ohne unerlaubte Hilfe angefertigt habe,
- ich mich nicht bereits anderwärtig um einen Doktorgrad beworben habe oder einen solchen besitze, und
- mir die Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät II der Humboldt-Universität zu Berlin bekannt ist, gemäß Amtliches Mitteilungsblatt Nr. 34/2006.

Berlin, den 26.5.2010 Siamak Tazari

Well, my heart's in the Highlands at the break of day
Over the hills and far away
There's a way to get there and I'll figure it out somehow
But I'm already there in my mind
And that's good enough for now

– Bob Dylan, 1997