Designing Subexponential Algorithms: Problems, Techniques & Structures

Frederic Dorn

Thesis for the degree of Philosophiae Doctor (PhD)
Department of Informatics

UNIVERSITY OF BERGEN

Bergen, Norway
July 2007

Abstract

In this thesis we focus on *subexponential algorithms* for NP-hard graph problems: exact and parameterized algorithms that have a *truly* subexponential running time behavior. For input instances of size n we study exact algorithms with running time $2^{O(\sqrt{n})}$ and parameterized algorithms with running time $2^{O(\sqrt{k})} \cdot n^{O(1)}$ with parameter k, respectively. We study a class of problems for which we design such algorithms for three different types of graph classes: planar graphs, graphs of bounded genus, and H-minor-free graphs. We distinguish between *unconnected* and *connected* problems, and discuss how to conceive parameterized and exact algorithms for such problems.

We improve upon existing dynamic programming techniques used in algorithms solving those problems. We compare tree-decomposition and branch-decomposition based dynamic programming algorithms and show how to unify both algorithms to one single algorithm. Then we give a dynamic programming technique that reduces much of the computation involved to fast matrix multiplication. In this manner, we obtain branch-decomposition based algorithms on numerous problems, such as Vertex Cover and Dominating Set. We also show how to exploit planarity for obtaining faster dynamic programming approaches, *a*) in connection with fast matrix multiplication and *b*) for tree-decompositions.

Furthermore, we focus on *connected* problems in particular, and their relation to the input graph structure. We state the basis for how the latter problems can be attacked for graph classes that inherit the *Catalan structure*. Truly subexponential algorithms for edge-subset problems such as k-Longest Path and Planar Graph TSP are derived by employing the planar graph structure. Moreover, we investigate how to obtain truly subexponential algorithms for torus-embedded graphs, bounded genus graph and H-minor-free graphs, by first using planarization techniques, and then proving the Catalan structure for the planarized instances.

Preface "So Long, and Thanks for All the Fish"

All work on this thesis has been funded mainly by *Exact Algorithms for Hard Problems* NFR FRINAT grant and additionally by the L. Meltzer Høyskolefond.

My first and sincere thanks go to my supervisor Professor Fedor Fomin for his wise guidance. Without his foresight, patience and trust in me, I might not have come to this point.

Further, I send my warmest thanks to all the members (and former members) of the algorithms group in Bergen (in alphabetical order), with whom I was sharing a great and inspiring working atmosphere: Joanna Bauer, Lene Favrholdt, Serge Gaspers, Petr Golovach, Pinar Heggernes, Kjartan Høie, Daniel Lokshtanov, Federico Mancini, Fredrik Manne, Daniel Meister, Rodica Mihai, Morten Mjelde, Charis Papadopoulos, Artem Pyatkin, Saket Saurabh, Christian Sloper, Alexey Stepanov, Jan Arne Telle, Yngve Villanger, and Qin Xin.

Many thanks for the research invitations and hospitality of Hans Bodlaender, Andrzej Lingas, Bojan Mohar, Ulrike Stege, and Dimitrios Thilikos.

I also take the opportunity to warmly thank Hans Bodlaender, Eelko Penninkx, Jan Arne Telle, and Dimitrios Thilikos for collaborating on the results of this thesis and Jochen Alber, Matt DeVos, Eva-Marta Lundell, Bojan Mohar, Rolf Niedermeier, and Ulrike Stege for other collaborations. Also, I want to thank the European Association for Theoretical Computer Science (EATCS) for giving me the honor of receiving their award.

Last but not least, I want to thank all my friends, present and past, for all your love and support through hard and good times.

This work is dedicated to my sister Nathalie Dorn, who has always been my greatest support—*danke!*

The thesis is based on the results of the following works:

- F. Dorn, *Dynamic programming and fast matrix multiplication*, in Proceedings of the fourteenth Annual European Symposium on Algorithms (ESA 2006), vol. 4168 of LNCS, Springer, 2006, pp. 280–291.
- F. Dorn, *How to use planarity efficiently: new tree-decomposition based algorithms*, in Proceedings of the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2007), LNCS, Springer, 2007, p. to appear.
- F. Dorn, F. V. Fomin, and D. M. Thilikos, *Fast subexponential algorithm for non-local problems on graphs of bounded genus*, in Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006), vol. 4059 of LNCS, Springer, 2006, pp. 172–183.
- F. Dorn, F. V. Fomin, and D. M. Thilikos, *Catalan structures and dynamic programming on H-minor-free graphs*, submitted, (2007).

- F. Dorn, F. V. Fomin, and D. M. Thilikos, *Subexponential parameterized algorithms*, in Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP 2007), vol. 4596 of LNCS, Springer, 2007, pp.15–27.
- F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin, *Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions*, submitted (2006).
- F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin, *Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions*, in Proceedings of the thirteenth Annual European Symposium on Algorithms (ESA 2005), vol. 3669 of LNCS, Springer, 2005, pp. 95–106.
- F. Dorn and J. A. Telle, *Semi-nice tree-decompositions: the best of branchwidth, treewidth and pathwidth with one algorithm*, submitted, (2005).
- F. Dorn and J. A. Telle, *Two birds with one stone: the best of branchwidth and treewidth with one algorithm*, in Proceedings of the seventh Latin American Theoretical Informatics Symposium (LATIN'06), vol. 3887 of LNCS, Springer, 2006, pp. 386–397.

This work includes only results to which I contributed in significant manner and to whose achievement I played a major role.

Bergen, 19.7.07 Frederic Dorn

Contents

1 Introduction

1.1 Subexponential algorithms: on the border between P and NP

Algorithms have many applications. Every software program, every machine or robot program is based on some algorithm, sophisticated or simple, involved or straightforward. Most of the implemented algorithms have been only developed for the problem-specific purpose and often in a non-optimal way. Though algorithmic problems form the heart of computer science, they hardly come as mathematically precise questions. Whereas the hardware and applications are developing with light-speed over the few last decades, the theory of algorithms can barely keep up the pace. This has the result that the gap between theory and application increases dramatically. For example, it is daily practice to solve NP-hard problems efficiently even though from the theoretical point of view it should not be possible! Exactly here, at the borderline between "efficient" and "inefficient" algorithms, the subject of this work is settled.

A main tool for algorithms theory, serving as a language for algorithms, is *graph theory*. A graph is a strictly algebraic object: a graph G is defined by a tuple of a set of *vertices* $V(G)$ and a set of *edges* $E(G) \subseteq [V(G)]^2$. We can visualize the vertices of a graph as a set of points with some connected by arcs–the edges. Most computational problems can be expressed as graph problems. The advantage is obvious: graphs reveal the core of the problem and help to find intuitive methods. For example, a navigation program stores a road map as a graph with weighted edges. For calculating the distance between two points, the program must solve the *shortest path* routine, which has its power from a long line of research on algorithmic graph theory.

Illuminating different aspects of algorithmic graph theory, this work is focused on the design and analysis of subexponential exact and parameterized algorithms for hard problems.

The widely believed conjecture is that exponential time algorithms are the best we can hope for solving NP-hard problems exactly. But for many NP-hard problems arising from logistics, molecular biology, networks etc., one can find algorithms that are useful up to a practically remarkable input size. The idea is that the input always comes with a certain structure. In logistics, for example, often the input obeys the euclidean metric; in big networks there are always a few vertices that have big degree, the "server nodes"; In molecular biology all the treated graphs have the structure of tree-like graphs (molecules, genealogy) or chordal graphs (physical mapping of DNA). Thus, in the recent years, it has become more and more popular to study algorithms that have a so-called "subexponential" running time behavior. The goal is to exploit the structure of the input graph in order to achieve an exponent in the running time that is sublinear, in

our case a square root of the input size. In real-world and experimental instances the algorithm runs even much faster than the theoretical bounds suggest [4].

This work is focused on two aspects for finding *subexponential algorithms*. The "theoretic aspect" is where we focus on how to employ graph structures for solving problems with two kinds of subexponential algorithms: *parameterized* and *exact*. In the "algorithmic aspect" we examine and improve several *dynamic programming* techniques that can be used to obtain fast and efficient algorithms solving various problems.

1.1.1 Theoretic aspects: parameterized and exact algorithms

Parameterized algorithms. The theory of fixed-parameter algorithms and parameterized complexity has been thoroughly developed during the last two decades; see e.g. the books [41, 45, 75]. Usually, parameterizing a problem on graphs is to consider its input as a pair consisting of the graph G itself and a parameter k. Typical examples of such parameters are the size of a vertex cover, the length of a path or the size of a dominating set. Roughly speaking, a parameterized problem in graphs with parameter k is *fixed parameter tractable* if there is an algorithm solving the problem in $f(k) \cdot n^{O(1)}$ steps for some function f that depends only on the parameter.

Many NP-complete graph problems turn out to be fixed parameter tractable when restricted to the class of graphs of bounded treewidth (branchwidth). A common algorithmic technique for problems, asking for the existence of vertex/edge subsets of size k with certain properties, is based on branchwidth (treewidth) and involves the following two ingredients: The first is a combinatorial proof that, if the branchwidth (treewidth) of the input graph is at least $f(k)$ (where f is some function of k), then the answer to the problem is directly implied. The second is a $g(\mathbf{w}(G)) \cdot n^{O(1)}$ step dynamic programming algorithm for the problem (here $\mathbf{w}(G)$ is the branchwidth (treewidth) of the input graph G and g is a function only dependent on $\mathbf{w}(G)$). Such an algorithm exists due to Courcelle's Theorem on the relation between *monadic second-order logic (MSOL)* and treewidth (branchwidth) [26]. Monadic second-order logic is the extension of first-order logic when allowing quantifiers over sets. The theorem says that problems that are expressed in MSOL can be solved in linear-time on a given tree-decomposition of constant width. Though the theorem is a powerful classification tool, an efficient application suffers from huge constants in $g(\mathbf{w}(G))$, that is, g is an enormously growing exponential function.However, for problems such as Dominating Set, there are $c^{\mathbf{w}(G)} \cdot n^{O(1)}$ time dynamic programming based algorithms with small constants c (see e.g. [92]).

While there is strong evidence that most of fixed-parameter algorithms cannot have a running time of $2^{o(k)} \cdot n^{O(1)}$ (see [61, 19, 45]), for planar graphs it is possible to design subexponential parameterized algorithms with running times of the type $2^{O(\sqrt{k})} \cdot n^{O(1)}$ (see [28, 19] for further lower bounds on planar graphs) as e.g., for Planar k-Vertex Cover and Planar k-Dominating Set. For obtaining such a $2^{O(\sqrt{k})} \cdot n^{O(1)}$ step algorithm, we further require that **(a)** $f(k) = O(\sqrt{k})$ and **(b)** $g(k) = 2^{O(\mathbf{w}(G))}$. For planar graphs (and also for H-minor free graphs – see [34]) **(a)** can be proved systematically

using the idea of Bidimensionality [33]. However, not an equally general theory exists for **(b)**.

Since the first paper in this area appeared [3], the study of fast subexponential algorithms attracted a lot of attention. In fact, it not only offered a good ground for the development of parameterized algorithms, but it also prompted combinatorial results, of independent interest, on the structure of several parameters in sparse graph classes such as planar graphs [2, 5, 6, 21, 30, 44, 49, 57, 63], bounded genus graphs [31, 47], graphs excluding some single-crossing graph as a minor [38], apex-minor-free graphs [29] and H-minor-free graphs [31, 34, 33].

k-Longest Path. The research on another type of parameterized problems is motivated by the seminal result of Alon, Yuster, and Zwick in [10] that proved that a path of length $\log n$ can be found in polynomial time, answering to a question by Papadimitriou and Yannakakis in [76]. One of the open questions left in [10] was: "*Is there a polynomial time (deterministic or randomized) algorithm for deciding if a given graph G contains a path of length, say,* $\log^2 n$*?* ". Of course, a $2^{O(\sqrt{k})} \cdot n^{O(1)}$ step algorithm for checking if a graph contains a path of length k would resolve this question.

The bad news is that, for many combinatorial problems, a general algorithm of time $2^{O(\sqrt{k})} \cdot n^{O(1)}$ is missing. k-Longest Path is a typical example of such a problem. Here the certificate of a solution should satisfy a global connectivity requirement. For this reason, the dynamic programming algorithm must keep track of all the ways the required path may traverse the corresponding separator of the decomposition, that is a $\Omega(\ell^\ell)$ on the size ℓ of the separator and therefore of the treewidth/branchwidth. The same problem in designing dynamic programming algorithms appears for many other combinatorial problems in NP whose solution certificates are restricted by global properties such as connectivity. Other examples of such problems are Longest Cycle, Connected Dominating Set, Feedback Vertex Set, Hamiltonian Cycle and Graph Metric TSP.

Exact algorithms. Very close related to parameterized algorithms are exact algorithms. After the omnipresent conjecture that P unequals NP, we have the *exponential time hypothesis (ETH)* by Impagliazzo et al [61]. A major question of *exponential time complexity* is if 3-SAT is solvable in time $2^{o(n)}$. ETH, the conjecture that it is not, can be deduced for many other problems, such as that Independent Set is not solvable in time $2^{o(n)}$. For planar graphs (and other sparse graph classes) though, it is possible to design subexponential algorithms of running time $2^{O(\sqrt{n})}$ for problems such as Planar Independent Set, Planar Dominating Set [6, 48].

1.1.2 Algorithmic aspects: dynamic programming.

Dynamic programming is a common tool for solving NP-hard problems. For our problems, we consider dynamic programming, which is a method for reducing the runtime of algorithms exhibiting the properties of overlapping subproblems and optimal substructure. A standard approach for getting fast exact algorithms for computational problems is to apply dynamic programming across subsets of the solution space. Famous applications of dynamic programming are, among others, Dijkstra's algorithm Single Source and Destination Shortest Path algorithm, Bellman-Ford algorithm, the TSP problem, the Knapsack problem, Chain Matrix Multiplication and many string algorithms including the Longest-Common Subsequence problem. See [25] for an introduction to dynamic programming.

Tree- and branch-decompositions. The tree- (branch-) decomposition characterizes how tree-like a graph is and provides the graph parameter treewidth (branchwidth) $\mathbf{w}(G)$ as a measure of this tree-likeness. Although computing the treewidth (branchwidth) of a graph is NP-complete in general, Bodlaender [15] gave a linear-time FPT for treewidth being the parameter (and for branchwidth [17]). For planar graphs, the two notions have yet a different status: whereas it is a long outstanding open problem if the treewidth of planar graphs can be computed in polynomial time, Seymour and Thomas [89] proved planar branchwidth being in P and gave an efficient constructive algorithm.

For parameterized problems, tree- (branch-)decomposition based algorithms typically rely on a dynamic programming strategy with the combinatorial explosion in the runtime restricted to the small $\mathbf{w}(G)$ in the exponent.

Dynamic programming along either a branch-decomposition or a tree-decomposition of a graph both share the property of traversing a tree bottom-up and combining tables of solutions to problems on certain subgraphs that overlap in a bounded-size separator of the original graph.

To be more concrete, the dynamic programming algorithms we are interested in have a running time

$$c^{\mathbf{w}(G)} \cdot n^{O(1)},$$

where n is the cardinality of $V(G)$ and c is a small constant. A major interest in improving such existing algorithms is to minimize c in the base. Telle and Proskurowski [92] gave an algorithm based on tree-decompositions having width ℓ that computes the Dominating Set of a graph in time $O(9^{\mathbf{w}(G)}) \cdot n^{O(1)}$. Alber et al. [2] not only improved this bound to $O(4^{\mathbf{w}(G)}) \cdot n^{O(1)}$ by using several tricks, but also were the first to give a subexponential fixed parameter algorithm for Planar Dominating Set.

Planarity and dynamic programming. Many exact algorithms for problems on planar graphs, that are obtain by the Lipton-Tarjan planar separator theorem [71], usually

have running time $2^{O(\sqrt{n})}$ or $2^{O(\sqrt{n}\log n)}$, however the constants hidden in big-Oh of the exponent are a serious obstacle for practical implementation.

Recently there have been several papers [46, 22, 47, 48], showing that for planar graphs or graphs of bounded genus the base of the exponent in the running time of these algorithms could be improved by instead doing dynamic programming along a branch decomposition of optimal branchwidth—both notions are closely related to tree decomposition and treewidth. Fomin and Thilikos [46] significantly improved the result of [2] for Planar k-Dominating Set to $O(2^{15.13\sqrt{k}}k + n^3)$ where k is the size of the solution. The same authors [48] achieve small constants in the running time of a branch decomposition based exact algorithms for Planar Independent Set and Planar Dominating Set, namely $O(2^{3.182\sqrt{n}})$ and $O(2^{5.043\sqrt{n}})$, respectively.

Fast matrix multiplication. Numerous problems are solved by matrix multiplication, e.g. see [59] for computing minimal triangulations, [69] for finding different types of subgraphs as for example clique cut sets, and [25] for LUP-decompositions, computing the determinant, matrix inversion and transitive closure, to only name a few.

However, for NP-hard problems the common approaches do not involve fast matrix multiplication. Williams [97] established new connections between fast matrix multiplication and hard problems. He reduces the instances of the well-known problems Max-2-SAT and Max-Cut to exponential size graphs dependent on some parameter k, arguing that the optimum weight k-clique corresponds to an optimum solution to the original problem instance.

The idea of applying fast matrix multiplication is basically to use the information stored in the adjacency matrix of a graph in order to fast detect special subgraphs such as shortest paths, small cliques—as in the previous example—or fixed sized induced subgraphs. Uncommonly—as in [97]—we do not use the technique on a matrix representation of the input graph directly. Instead, it facilitates a fast search in the solution space. In the literature, there has been some approaches speeding up linear programming using fast matrix multiplication, e.g. see [94].

1.1.3 Our contribution

In algorithms theory one has typically two aims: either classifying a problem according two its complexity class, or improving upon the running time of existing algorithms. In the first case, the landscape of complexity theory has become more subtle in the recent decades. In the beginning the interest in a problem used to come to a halt as soon as it was allocated to either P or NP. Since then, the measurements for *tractable* and *intractable* have changed drastically (e.g. [52]), and one has started to incorporate the structure of the input instances to find new classification methods and complexity classes, such as for example the aforementioned FPT (e.g. [41]). Originally, the goal was to prove the existence of a parameterized algorithm of running time $f(k) \cdot n^{O(1)}$, and

there was little interest in finding a tractable function $f(k)$ for parameter k. In this work we employ the structure of the input graphs, given by their topological embedding, in order to classify the studied problems as *truly subexponential*. I.e., we give algorithms of running time $2^{O(\sqrt{n})}$ and $2^{O(\sqrt{k})} \cdot n^{O(1)}$ exact and parameterized, respectively, improving upon existing algorithms of running time $2^{O(\sqrt{n}\log n)}$ and $2^{O(\sqrt{k}\log k)} \cdot n^{O(1)}$.

The other goal in algorithms theory is to reduce the constant in the exponent of the running time of existing algorithms. Traditionally, due to practical interest, the focus has been on algorithms in P, e.g., improvement for $O(n^2)$ to $O(n^{1.5})$. However, with the increasing computational power, previously unfeasible problems, such as problems with subexponential algorithms have become practically solvable up to remarkable input sizes—by all means only if the corresponding constants were small enough (e.g. [4]). This goal—the finding of small constants—we stress in the other part of this thesis. In this line of research it turned out that for many such problems, dynamic programming is currently the most efficient tool to provide small constants (used e.g. in [2, 49]). We introduce new generic techniques that employ data structures, data representations and input structures, with the one goal—to give algorithms with practically applicable running time behavior.

Problems

After giving some notions in Section 1.2, we introduce the problems that are subject to this work in Section 1.3. We classify them as *vertex-subset problems* and *edge-subset problems*. Whereas, some of the vertex-subset problems we study, have been classified in literature, e.g. as (σ, ϱ)-problems [92], we give here a more general problem description, that we extend to edge-subset problems.

Vertex- and edge-subset problems. Vertex-subset problems ask for a subset of graph vertices as solution which has *local properties*. That is, for a subset of vertices one can verify non-deterministically for each vertex and its neighborhood if the subset is satisfying the problem as a (maybe non-optimal) solution. Edge-subset problems differ not only by asking for an edge subset as solution but also by the global property they own. We have to consider the entire subgraph induced by the edge subset for verifying if it is a correct solution.

Parameterized and exact algorithms. To give a survey and background on the thesis related works, we study general approaches for obtaining subexponential exact and parameterized algorithms for vertex- and edge-subset problems in Section 1.4 and we reveal their relation with combinatorial results related to the Graph Minors project. All these algorithms exploit the structure of graph classes that exclude some graph as a minor. This was used to develop techniques as Bidimensionality theory for better bounding the steps of dynamic programming when applied to minor closed graph classes.

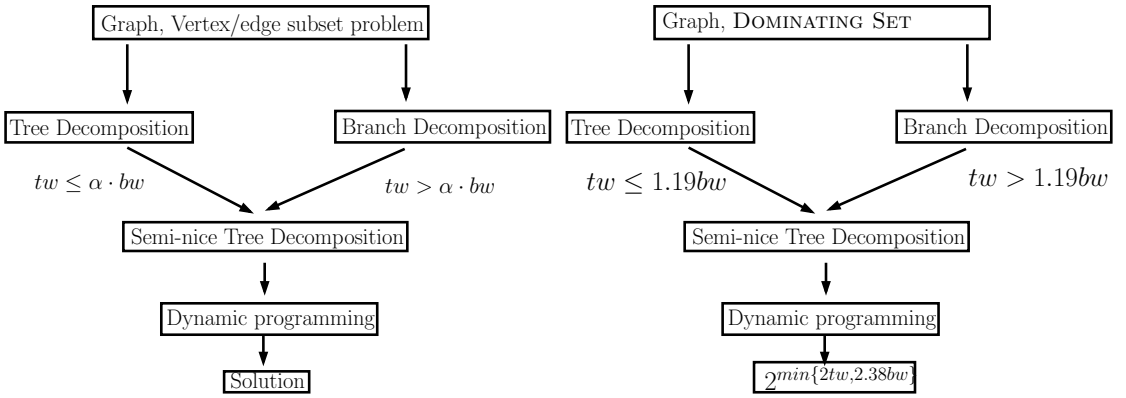


Figure 1.1: A single dynamic programming approach. α marks the trade-off parameter, depending on the given problem, on which we decide the decomposition type to transform. On the right, we have the example of Dominating Set with $\alpha = 1.16$.

Techniques

Branchwidth and treewidth are connectivity parameters of graphs of high importance in algorithm design. By dynamic programming along the associated branch- or tree-decomposition one can solve most graph optimization problems in time linear in the graph size and exponential in the parameter. If one of these parameters is bounded on a class of graphs, then so is the other, but they can differ by a small constant factor and this difference can be crucial for the resulting runtime.

Dynamic programming on tree-decompositions and branch-decompositions. In Chapter 2, we improve the dynamic programming techniques on both decompositions. After describing the common ways how fast dynamic programming algorithms on tree- and on branch-decompositions are obtained, we introduce semi-nice tree-decompositions and show that they combine the best of both branchwidth and treewidth. We first give simple algorithms to transform a given tree-decomposition or branch-decomposition into a semi-nice tree-decomposition. We then give a template for dynamic programming along a semi-nice tree-decomposition for optimization problems over vertex subsets. See Figure 1.1.

Dynamic programming and fast matrix multiplication. We give a new technique for combining dynamic programming and matrix multiplication and apply this approach to problems like Dominating Set and Vertex Cover for improving the best algorithms on graphs of bounded branchwidth. We introduce the new dynamic programming approach on branch decompositions. Instead of using tables, it stores the solutions in matrices that are computed via distance product. Since distance product is not known to have a fast matrix multiplication in general, we only consider unweighted and small

integer weighted problems with weights of size $M = n^{O(1)}$. Our approach is fully general. It runs faster than the usual dynamic programming for any vertex subset problem on graphs of bounded branchwidth.

We first introduce our idea by showing a straightforward way to solve Max Cut on general graphs. We then extend this idea to dynamic programming to obtain a technique on the Vertex Cover problem on graphs of branchwidth **bw** and show the improvement from $O(2^{1.5\,\mathbf{bw}}) \cdot n^{O(1)}$ to $O(M \cdot 2^{\frac{\omega}{2}\,\mathbf{bw}}) \cdot n^{O(1)}$ where ω is the exponent of fast matrix multiplication (currently $\omega < 2.376$).

Next, we give the general technique and show how to apply it to several optimization problems such as Dominating Set, whose existing algorithm we improve from $O(3^{1.5\,\mathbf{bw}}) \cdot n^{O(1)}$ to $O(M \cdot 4^{\mathbf{bw}}) \cdot n^{O(1)}$.

Planarity and dynamic programming. We analyze the deep results of Seymour & Thomas [89] and extract geometric properties of planar branch decompositions. Loosely speaking, their results imply that for a graph G embedded on a sphere $\mathbb{S}_0$, some branch decompositions can be seen as decompositions of $\mathbb{S}_0$ into discs (or sphere cuts). We describe such geometric properties of so-called *sphere-cut decompositions*. Sphere-cut decompositions seem to be an appropriate tool for solving a variety of planar graph problems, because next to being computable in polynomial time, their geometric properties can be used to improve upon algorithmic techniques.

In combination with fast matrix multiplication, we show the significant improvement of the constants of the runtime for the approach on planar graph problems. On Planar Dominating Set we reduce the time to $O(M \cdot 2^{0.793\omega\,\mathbf{bw}}) \cdot n^{O(1)}$ and hence an improvement of the fixed parameter algorithm in [46] to $O(2^{11.98\sqrt{k}}) \cdot n^{O(1)}$ where k is the size of the dominating set. For exact subexponential algorithms as on Planar Independent Set and Planar Dominating Set, this means an improvement to $O(M \cdot 2^{1.06\omega\sqrt{n}})$ and $O(M \cdot 2^{1.679\omega\sqrt{n}})$, respectively.

In contrast to branch-decompositions, tree-decompositions have been historically the choice when solving NP-hard optimization and FPT problems with a dynamic programming approach (see for example [13] for an overview). Although much is known about the combinatorial structure of tree-decompositions (e.g., [14, 95]), only few results are known to the author relating to the topology of tree-decompositions of planar graphs (e.g., [18]). We give the first result which employs planarity obtained by the structure of tree-decompositions for getting faster algorithms. Exploiting planarity, we improve further upon the existing bounds and give a $3^{\mathbf{tw}} \cdot n^{O(1)}$ algorithm for Planar Dominating Set, representative for a number of improvements on results of [7]See Figure 1.2 for an overview on the results on dynamic programming presented in this work.

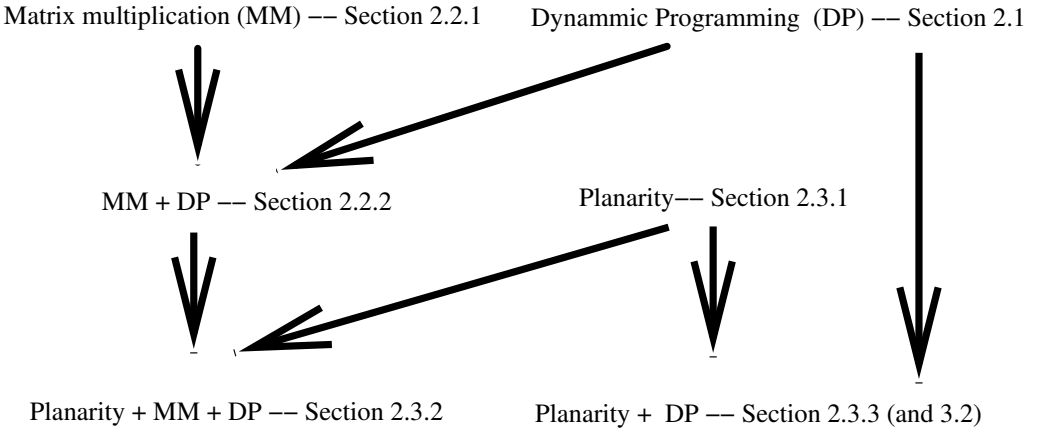


Figure 1.2: The connections between the different techniques on dynamic programming investigated in Chapter 2.

Structures

Connectivity and surfaces. Many separator results for topological graphs, especially for planar embedded graphs base on the following fact: Separators have a structure that cuts the surface into two or more pieces onto which the separated subgraphs are embedded on. The celebrated and widely applied (e.g., in many divide-and-conquer approaches) result of Lipton and Tarjan [71] finds in planar graphs a small sized separator. However, their result says nothing about the structure of the separator, it can be any set of discrete points. Applying the idea of Miller for finding small simple cyclic separators [73] in planar triangulations, one can find small separators whose vertices can be connected by a closed curve in the plane intersecting the graph only in vertices, so-called *Jordan curves* (e.g. see [11]). This permits to view collections of paths that may pass through such a separator as non-crossing pairings of the vertices of a cycle (Section 3.1). We consider sphere-cut decompositions of the input graph, whose separator form Jordan curves, and observe that the number of ways a path traverses a separator of the decomposition is linearly bounded by the Catalan number of the separator size . We use such decompositions with the Catalan structure in Chapter 3 for improving algorithms for edge subset problems. We also give the basis for finding such decompositions with the Catalan structure for solving edge subset problems on graphs of bounded genus and H-minor-free graphs.

Planar graphs. In Section 3.2, we study a new framework for designing fast subexponential exact and parameterized algorithms on planar graphs for edge-subset problems. The approach is based on geometric properties of planar branch decompositions obtained by Seymour & Thomas, combined with refined techniques of dynamic programming on planar graphs based on properties of non-crossing partitions. Compared to divide-and-conquer algorithms, the main advantages of our method are a) it is a generic method which allows to attack broad classes of problems; b) the obtained algorithms

provide a better worst case analysis. To exemplify our approach we show how to obtain an $O(2^{10.984\sqrt{k}}n^{3/2} + n^3)$ time algorithm solving k-Longest Path using above fast matrix multiplication. We also show how our technique can be used to solve Planar Graph TSP in time $O(2^{8.15\sqrt{n}})$. We also employ the structure of planar graphs in order to obtain new combinatorial results on other widely studied problems, such as Hamiltonicity, Steiner Tree, Feedback Vertex Set and Max Leaf Tree. Thereby, we are improving the runtime of the best formerly known algorithms by a logarithmic factor in the exponent.

Graphs of bounded genus. Topological graph theory is an area of graph theory that studies the embedding of graphs on surfaces, and graphs as topological spaces. Many results, overall the separator theorem, can be extended from planar graphs to graphs of higher genus [54], that are graphs embedded crossing-free on surfaces of higher genus. However, this is not always a straightforward task. We investigate a general technique to design fast subexponential algorithms for graphs of bounded genus for edge-subset problems. Our algorithms are "fast" in the sense that they avoid the $\log k$ $(\log n)$ overhead and also because the constants hidden in the big-Oh of the exponents are reasonable. The technique we use is based on reduction of the bounded genus instances of the problem to *planar* instances of a more general graph problem on planar graphs where Catalan structure arguments are still possible. Such a reduction employs several results from topological graph theory concerning graph structure and noncontractible cycles of non-planar embeddings.

In Section 3.3, we first exemplify how to obtain a fast subexponential algorithm for the Hamiltonian Cycle problem on torus-embedded graphs. These graphs already inherit all "nasty" properties of non-planar graphs and all difficulties arising on surfaces of higher genus appear for torus-embedded graphs. However, the case of torus-embedded graphs is still sufficiently simple to exemplify the minimization technique used to obtained reasonable constants in the exponent. Hereafter, we explain in Section 3.4 how the results on torus-embedded graphs can be extended for any graphs embedded in a surface of fixed genus.

H-minor-free graphs. The likely most important work in recent graph theory is the Graph Minor Theory developed by Robertson and Seymour in a series of over 20 papers spanning over 20 years. The core of the Graph Minor Theory is a structure theorem [85] capturing the structure of all graphs excluding a fixed minor, also called H-minor-free graphs. That is, the theorem says that every such graph can be decomposed into a collection of graphs, each of which can "almost" be embedded into a surface of bounded genus, and combined in a tree-like structure. The structure theorem has been employed in order to conceive polynomial time approximation schemes and subexponential algorithms for problems restricted to a class of graphs with an excluded minor (see e.g. [55, 31, 34, 33]). In Section 3.5,we study how to exploit this structure in a way that edge-subset problems can be solved efficiently in subexponential running time on H-minor-free graphs. We

devise a $2^{O(\sqrt{k})} \cdot n^{O(1)}$ step algorithm for the k-Longest Path problem on H-minor free graphs, implying a polynomial-time algorithm for a $\log^2 n$-length path. This result is tight, because, according to Deĭneko, Klinz, and Woeginger [28], the existence of a $2^{o(\sqrt{k})} \cdot n^{O(1)}$ step algorithm, even for planar graphs, would again violate the exponential time hypothesis.

The main combinatorial result we use, concerns the existence of suitably structured branch decompositions of H-minor free graphs. While the grid excluding part follows directly from [34], the construction of this branch decomposition is quite involved. In fact, it uses the factthat any H-minor free graph can roughly be obtained by identifying in a tree-like way small cliques of a collection of components that are almost embeddable on bounded genus surfaces. A constructive polynomial time algorithm is to be found in [36] (see also [27]). The main proof idea is based on a procedure of "almost"-planarizing the graphs almost embeddable on the surface of bounded genus. However, we require a planarizing set with certain topological properties, able to reduce the high genus embeddings to planar ones where the planarizing vertices are cyclically arranged in the plain. This makes it possible to use sphere cut decompositions with Catalan structures. This decomposition is used to build a decomposition on the initial almost embeddable graph. Then using the tree-like way these components are linked together, we build a branch decomposition of the entire graph. The most technical part of the proof is to prove that each step of this construction, from the almost planar case to the entire graph, maintains the Catalan structure.

1.2 Notions

In this work, the main objects of our study are on the one hand graphs and sparse graph classes, in particular, planar graphs, graphs of bounded genus, and H-minor-free graphs. On the other hand, we also study graph related structures, such as branch decompositions and tree decompositions. Throughout this work, we use the following notions.

1.2.1 Graphs

If not otherwise stated, we deal with undirected graphs without loops and multiple edges. A *graph* G is a tuple consisting of a *vertex* set $V(G)$ and end *edge* set $E(G) \subseteq [V(G)]^2$. That is, the elements of $E(G)$ are two-element subsets of $V(G)$. We use n to denote the size of the vertex set and m for the size of the edge set, respectively. The graph G' is a *subgraph* of graph G if $V' \subseteq V(G)$ and $E' \subseteq E(G)$. If $E' = E(G) \cap [V']^2$, we also call G' an *induced subgraph* of G and denote it by $G[V']$.

For a vertex $v \in V(G)$ we define the *neighborhood* $N(v) = \{u \in V(G) : \exists\{u, v\} \in E(G)\}$ to be the set of vertices adjacent to v. We define the *degree* of vertex v by $|N(v)|$. For a subset $S \subseteq V(G)$ of the vertices of a graph G we let the neighborhood

$N(S) = \{v \notin S : \exists u \in S \wedge \{u, v\} \in E(G)\}$ be the set of vertices not in S that are adjacent to some vertex in S. A *path* between two vertices $x_1, x_\ell \in V(G)$ is a collection of vertices $x_1, \ldots, x_\ell \in V(G)$ such that $\{x_i, x_{i+1}\} \in E(G)$ for all $1 \leq i < \ell$ and $x_i \neq x_j$ for all $1 \leq i \neq j \leq \ell$. A graph is *connected* if there exists a path between any two distinct vertices $v, w \in V(G)$. A maximal connected subgraph of a graph is called *(connected) component*. A graph is a *clique* K_n if $E(K_n) = [V(K_n)]^2$. We call G_{n_1,n_2} a *bipartite graph* if $V(G_{n_1,n_2})$ can be partitioned into V_1 and V_2 with $n_i = |V_i|$ $(i = 1, 2)$ such that $E(G_{n_1,n_2}) \subseteq [V_1 \cdot V_2]$. K_{n_1,n_2} is *complete bipartite*, if $E(K_{n_1,n_2}) = [V_1 \cdot V_2]$. A $(r \times r)$*-grid* $G_{r,r}$ is the Cartesian product of two paths P_r of length r: $G_{r,r} = P_r \times P_r$.

Graph separators. Given a connected graph G, a set of vertices $S \subset V(G)$ is called a *separator* if the subgraph induced by $V(G) \setminus S$ has at least two components. S is called an u, v*-separator* for two vertices u and v that are in different components of $G[V(G) \setminus S]$. S is a *minimal* u, v*-separator* if no proper subset of S is a u, v-separator. Finally, S is a *minimal separator* of G if there are two vertices u, v such that S is a minimal u, v-separator. For a vertex subset $A \subseteq V(G)$, we *saturate* A by adding edges between every two non-adjacent vertices, and thus, turning A into a clique.

Graph operations. Given an edge $e = \{x, y\}$ of a graph G, the graph G/e is obtained from G by contracting the edge e; that is, to get G/e we identify the vertices x and y and remove all loops and replace all multiple edges by simple edges. A graph H obtained by a sequence of edge-contractions is said to be a *contraction* of G. H is a *minor* of G if H is a subgraph of a contraction of G. We use the notation $H \preceq G$ (resp. $H \preceq_c G$) when H is a minor (a contraction) of G. We say that a graph G is H*-minor-free* when it does not contain H as a minor. We also say that a graph class $\mathcal{G}$ is H*-minor-free* (or, excludes H as a minor) when all its members are H-minor-free. A very important graph class is the class of *planar graphs* which has a famous purely combinatorial characterization due to Kuratowski, namely the intersection of the K_5-minor-free and $K_{3,3}$-minor-free graph classes.

1.2.2 Width parameters

The graph parameters treewidth and branchwidth have been introduced by Robertson and Seymour in their seminal work on Graph Minors theory [81, 83] and since then played an important role in both, graph theory and algorithm theory.

Tree-decompositions. Let G be a graph, T a tree, and let $\mathcal{Z} = (Z_t)_{t \in T}$ be a family of vertex sets $Z_t \subseteq V(G)$, called *bags*, indexed by the nodes of T. The pair $\mathcal{T} = (T, \mathcal{Z})$ is called a *tree-decomposition* of G if it satisfies the following three conditions:

- $V(G) = \cup_{t \in T} Z_t$,
- for every edge $e \in E(G)$ there exists a $t \in T$ such that both ends of e are in Z_t,

- $Z_{t_1} \cap Z_{t_3} \subseteq Z_{t_2}$ whenever t_2 is a vertex of the path connecting t_1 and t_3 in T.

The width $\mathbf{tw}(\mathcal{T})$ of the tree-decomposition $\mathcal{T} = (T, \mathcal{Z})$ is the maximum size over all bags minus one.

Branch-decompositions. A *branch-decomposition* (T, μ) of a graph G consists of an ternary tree T (i.e. all internal vertices of degree three) and a bijection $\mu : L \to E(G)$ from the set L of leaves of T to the edge set of G. We define for every edge e of T the *middle set* $\operatorname{mid}(e) \subseteq V(G)$ as follows: Let T_1 and T_2 be the two connected components of $T \setminus \{e\}$. Then let G_i be the graph induced by the edge set $\{\mu(f) : f \in L \cap V(T_i)\}$ for $i \in \{1, 2\}$. The *middle set* is the intersection of the vertex sets of G_1 and G_2, i.e., $\operatorname{mid}(e) := V(G_1) \cap V(G_2)$. Whenever we are more interested in the graph induced by the edges E_e, that are all the preimages of the leaves of one of the connected components of $T \setminus e$, we eventually denote $\operatorname{mid}(e)$ as the vertex set ∂E_e. The *width* $\mathbf{bw}$ of (T, μ) is the maximum order of the middle sets over all edges of T, i.e., $\mathbf{bw}(T, \mu) := \max\{|\operatorname{mid}(e)| : e \in T\}$.

Tree- and branchwidth. For a graph G its treewidth $\mathbf{tw}(G)$ and branchwidth $\mathbf{bw}(G)$ is the smallest width of any tree-decomposition and branch-decomposition of G respectively. The pathwidth $\mathbf{pw}(G)$ is the smallest width of a tree-decomposition $(T, \mathcal{Z})$ where T is a path, a so-called *path decomposition*. The three graph parameters treewidth, branchwidth and pathwidth were introduced by Robertson and Seymour as tools in their seminal proof of the Graph Minors Theorem. The treewidth $\mathbf{tw}(G)$ and branchwidth $\mathbf{bw}(G)$ of a graph G satisfy the relation [80]

$$\mathbf{bw}(G) \leq \mathbf{tw}(G) + 1 \leq \frac{3}{2}\, \mathbf{bw}(G),$$

and thus whenever one of these parameters is bounded by some fixed constant on a class of graphs, then so is the other. Note that it is well known that $H \preceq G$ or $H \preceq_c G$ implies both, $\mathbf{bw}(H) \leq \mathbf{bw}(G)$ as well as $\mathbf{tw}(H) \leq \mathbf{tw}(G)$.

See Figure 1.3 for an illustration of tree-decompositions and branch-decompositions.

Trunk decompositions. We introduce a new kind of decomposition for exclusive usage in Section 3.5. If in the definition of a branch decomposition we further demand the ternary tree T to be a caterpillar, then we define the notion of a *trunk decomposition* and the parameter of the *trunkwidth* of a graph. For a longest path with edges $e_1, \ldots, e_q$ of such a caterpillar, the sets $X_i = \operatorname{mid}(e_i)$ form a linear ordering $\mathcal{X} = (X_1, \ldots, X_q)$. For convenience, we will use ordered sets to denote a trunk decompositions and, in order to include all vertices of G in the sets of $\mathcal{X}$, we will often consider trunk decompositions of $\hat{G}$ that is G with loops added to all its vertices (this operation cannot increase the width by more than one). The trunkwidth of a graph G is either $\mathbf{pw}(G) - 1$, $\mathbf{pw}(G)$, or $\mathbf{pw}(G) + 1$.

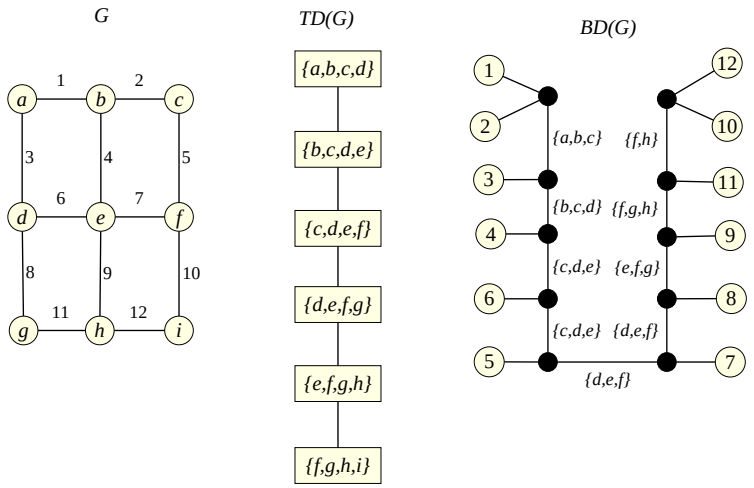


Figure 1.3: The left diagram shows a graph with vertex and edge labels. In the middle, we see a tree-decomposition of the graph on the left, and on the right a branch-decomposition.

1.2.3 Chordal graphs

A *chord* in a cycle C of a graph G is an edge joining two non-consecutive vertices of C. A graph H is called *chordal* if every cycle of length > 3 has a chord.

There exists an alternative characterization of chordal graphs:

Theorem 1.2.1. *[40] A graph H is chordal if and only if every minimal separator of H is a clique.*

Minimal triangulations. A *triangulation* of a graph G is a chordal graph H with $V(H) = V(G)$ and $E(G) \subseteq E(H)$. The edges of $E(H) \setminus E(G)$ are called *fill edges*. We say that H is a *minimal triangulation* of G if every graph G' with $V(G') = V(G)$ and $E(G) \subseteq E(G') \subset E(H)$ is not chordal. Note that a triangulation of a planar graph may not be planar—not to confuse with the notion of "planar triangulation" that asks for filling the facial cycles with chords. Consider the following algorithm that triangulates G, known as the *elimination game* [78]. Repeatedly choose a vertex, saturate its neighborhood, and delete it. Terminate when $V(G) = \varnothing$. The order in which the vertices are deleted is called the *elimination ordering* α, and G^+_α is the chordal graph obtained by adding all saturating (fill) edges to G. Another way of triangulating a graph G can be obtained by using a tree-decomposition of G.

If there is an ordering α such that no edges are added during the algorithm, i.e. $G = G^+_\alpha$, we say that G has a *perfect elimination ordering*, short *p.e.o.*.

Theorem 1.2.2. *[51] A graph is chordal if and only if it has a perfect elimination*

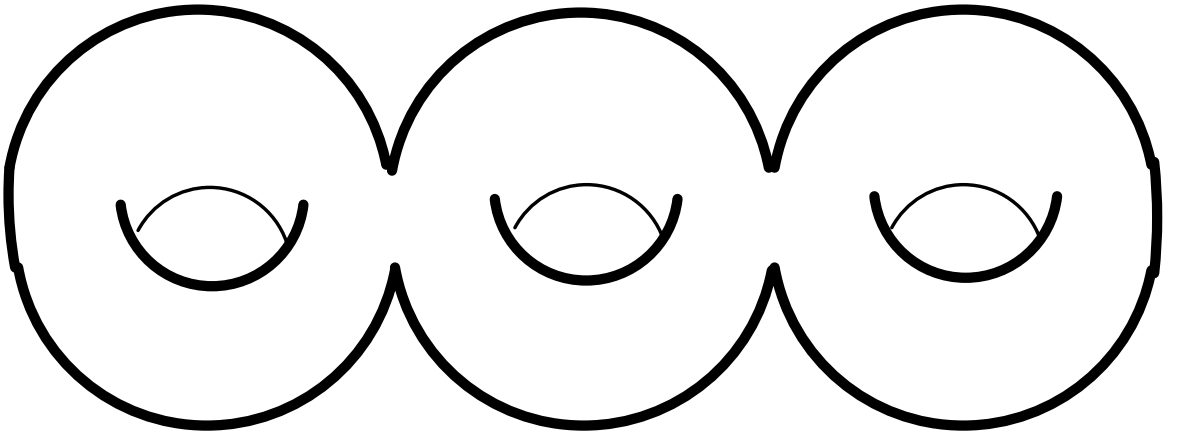


Figure 1.4: Surface $\mathbb{S}_3$ is obtained from the sphere by adding three handles.

ordering.

1.2.4 Graphs on surfaces

A *surface* Σ is a connected, compact 2-manifold without boundary (we always consider connected surfaces). We denote by $\mathbb{S}_0$ the sphere $(x, y, z \mid x^2 + y^2 + z^2 = 1)$ and by $\mathbb{S}_1$ the torus $(x, y, z \mid z^2 = 1/4 - (\sqrt{x^2 + y^2} - 1)^2)$. A *line* in Σ is a subset homeomorphic to $[0, 1]$. An O-arc is a subset of Σ homeomorphic to a circle. Let Δ be a closed disk and the open disk $\mathbf{int}(\Delta)$ its interior and $\mathbf{bor}(\Delta)$ its boundary. Then $\Delta = \mathbf{int}(\Delta) \cup \mathbf{bor}(\Delta)$.

Genus. Let Δ_1 and Δ_2 be two disjoint disks in Σ, and O_1, O_2 two disjoint O-arcs each boundary of Δ_1 and Δ_2, respectively. We form a new surface Σ' by deleting $\mathbf{int}(\Delta_1)$ and $\mathbf{int}(\Delta_2)$ from Σ and identifying O_1 with O_2 such that the clockwise orientations around O_1 and O_2 disagree. We say that we obtain Σ' from Σ by *adding a handle*. If we start with $\mathbb{S}_0$ and iteratively add h handles to it, we call the surface $\mathbb{S}_h$ that we obtain *orientable surface of genus* h. If we identify O_1 and O_2 with agreeing orientations, we obtain *crosscraps*. We call the surface $\mathbb{N}_h$ *nonorientable* if it is obtained from $\mathbb{S}_0$ by adding h crosscraps. See Figure 1.4 for an illustration.

Euler genus. The Euler genus of a surface Σ is $\mathbf{eg}(\Sigma) = \min\{2\mathbf{g}(\Sigma), \tilde{\mathbf{g}}(\Sigma)\}$ where $\mathbf{g}$ is the orientable genus and $\tilde{\mathbf{g}}$ the nonorientable genus of Σ. Whenever we talk about genus and surfaces from now on, we mean Euler genus and orientable surfaces.

Graph embedding. Whenever we refer to a Σ-*embedded graph* G with vertex set $\{v_1, v_2, \ldots, v_n\}$, we consider a 2-cell embedding of G in a surface Σ. That is a collection $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ such that π_i is a cyclic permutation of the edges incident with v_i for $i = 1, 2, \ldots, n$. We say that the embedding $\Pi' = \{\pi'_1, \pi'_2, \ldots, \pi'_n\}$ is the same

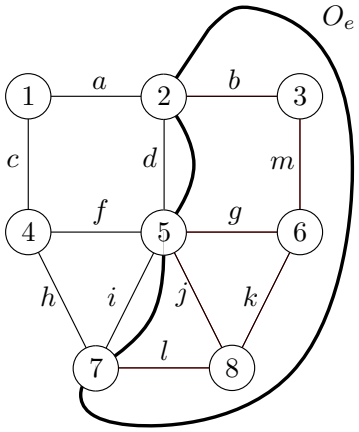


Figure 1.5: The diagram shows a plane graph which is separated by a noose two parts.

as Π if $\pi'_i = \pi_i$ for $i = 1, 2, \ldots, n$ or $\pi'_i = \pi_i^{-1}$ for $i = 1, 2, \ldots, n$. We call a $\mathbb{S}_0$-embedded graph together with its embedding a *plane graph*.

To simplify notations we do not distinguish between a vertex of G and the point of Σ used in the drawing to represent the vertex or between an edge and the line representing it. We also consider G as the union of the points corresponding to its vertices and edges. That way, a subgraph H of G can be seen as a graph H where $H \subseteq G$. We call by *region* of G any connected component of $(\Sigma \setminus E(G)) \setminus V(G)$. (Every region is an open set.) If $\mathbf{eg}(\Sigma)$ is a constant, we call a Σ-embedded graph also a bounded genus graph .

Radial graphs. Given a Σ-embedded graph G, its *radial graph* (also known as vertex-face graph) is defined as the the graph R_G that has as vertex set the vertices and the faces of G and where an edge exists if and only if it connects a face and a vertex incident to it in G (R_G is also a Σ-embedded graph).

Jordan curves / Nooses. A subset of Σ meeting the drawing only in vertices of G is called *G-normal*. If an O-arc is G-normal then we call it *noose* or, eventually, *(closed) Jordan curve*. The main differences between these two notions are that we sometimes consider Jordan curves without a given graph, and that we allow Jordan curves to not intersect with vertices at all. The length of a noose N is the number of its vertices and we denote it by $|N|$. If $\mathbf{int}(\Delta)$ is subset of a region of G, then $\mathbf{bor}(\Delta)$ is a noose. Two nooses are *homotopic* if there is a continuous deformation of one onto the other. A noose is *contractible* if it is homotopic to a single point. Otherwise, it is *noncontractible*. On a sphere $\mathbb{S}_0$, every noose N bounds two open discs Δ_1, Δ_2, i.e., $\Delta_1 \cap \Delta_2 = \varnothing$ and $\Delta_1 \cup \Delta_2 \cup N = \mathbb{S}_0$. See Figure 1.5 for an illustration.

Tight nooses. If the intersection of a noose with any region results into a connected subset, then we call such a noose *tight*. Notice that each tight noose N in a Σ-embedded

graph G, corresponds to some cycle C of its radial graph R_G (notice that the length of such a cycle is $2 \cdot |N|$). Also any cycle C of R_G is a tight noose in G.

Shortest noncontractible nooses. As it was shown by Thomassen in [93] (see also Theorem 4.3.2 of [74]) a shortest noncontractible cycle in a graph embedded on a surface can be found in polynomial time. By Proposition 5.5.4 of [74] a noncontractible noose of minimum size is always a tight noose, i.e. corresponds to a cycle of the radial graph. Thus we have the following proposition.

Proposition 1.2.3. *There exists a polynomial time algorithm that for a given Σ-embedded graph G, where $\Sigma \neq \mathbb{S}_0$, finds a noncontractible tight noose of minimum size.*

Representativity. Representativity [82] is the measure how dense a graph is embedded on a surface that is not a sphere. The *representativity* (or *face-width*) $\mathbf{rep}(G)$ of a graph G embedded in surface $\Sigma \neq \mathbb{S}_0$ is the smallest length of a noncontractible noose in Σ. In other words, $\mathbf{rep}(G)$ is the smallest number k such that Σ contains a noncontractible (non null-homotopic in Σ) closed curve that intersects G in k points. By Theorem 4.1 of [84], we have:

Lemma 1.2.4. *Given a Σ-embedded graph G where $\Sigma \neq \mathbb{S}_0$. Then*

$$\mathbf{rep}(G) \leq \mathbf{bw}(G).$$

Cutting nooses. Suppose that C is an oriented cycle of a Σ-embedded graph G. Let G' be the graph obtained from G by replacing C with two copies of C such that all edges on the left side of C incident to C are now incident to one copy of C and all edges on the right side of C are incident with the other copy of C. We say that G' is obtained from G by *cutting along* C. See Figure 1.6 for an example of cutting. If we cut along a noncontractible cycle of the radial graph of G, that is a noncontractible tight noose N of G, we say that we *cut along a noncontractible noose*. We thus *duplicate* the vertices of N. We call the copies N_X and N_Y of N *cut-nooses*. Note that cut-nooses are not necessarily tight (In other words, a cut-noose can enter and leave a region of G several times.) We can see the operation of "cutting G along a non-contractible noose N" as "sawing" the surface where G is embedded. This helps us to embed the resulting graph to the surface(s) that result after adding to the sawed surface two disks, one for each side of the splitting. We call these disks *holes* and we will treat them as closed disks. Clearly, in the new embedding(s) the duplicated vertices will all lay on the borders of these holes.

The following lemma is very useful in proofs by induction on the genus. The first part of the lemma follows from Proposition 4.2.1 (corresponding to surface separating cycle) and the second part follows from Lemma 4.2.4 (corresponding to non-separating cycle) in [74].

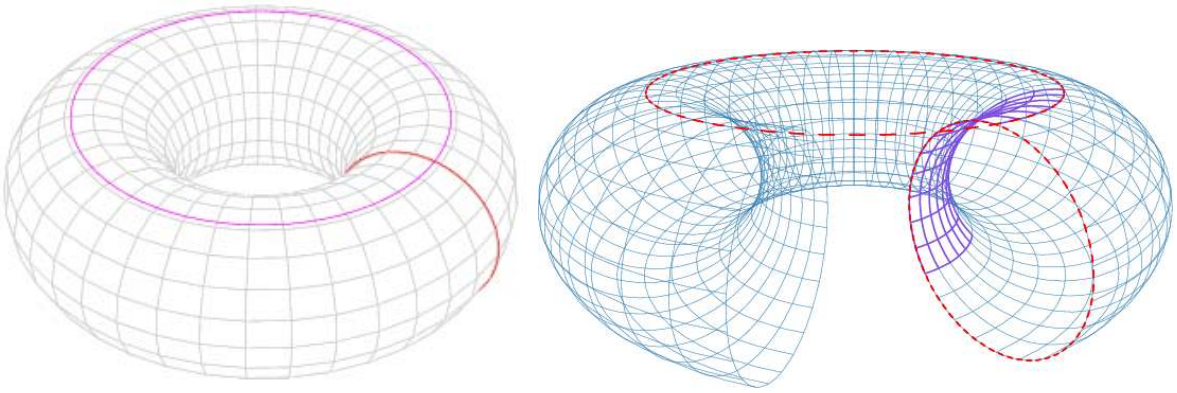


Figure 1.6: The left diagram shows grid graph embedded on a torus with two nonhomotopic noncontractible cycles. On the right, we see the grid after cutting along one such cycle. Note that this surface is a cylinder and the graph is now planar.

Proposition 1.2.5. *Let G be a Σ-embedded graph where $\Sigma \neq \mathbb{S}_0$ and let G' be a graph obtained from G by cutting along a noncontractible tight noose N on G. One of the following holds*

- *G' can be embedded in a surface with Euler genus strictly smaller than $\mathbf{eg}(\Sigma)$.*
- *G' is the disjoint union of graphs G_1 and G_2 that can be embedded in surfaces Σ_1 and Σ_2 such that $\mathbf{eg}(\Sigma) = \mathbf{eg}(\Sigma_1) + \mathbf{eg}(\Sigma_2)$ and $\mathbf{eg}(\Sigma_i) > 0$, $i = 1, 2$.*

Sphere-cut Decompositions. For a plane graph G, we define a *sphere-cut decomposition* or *sc-decomposition* (T, μ, π) as a branch decomposition such that for every edge e of T there exists a noose O_e bounding the two open discs Δ_1 and Δ_2 such that $G_i \subseteq \Delta_i \cup O_e$, $1 \leq i \leq 2$. Thus O_e meets G only in $\mathrm{mid}(e)$ and its length is $|\mathrm{mid}(e)|$. A clockwise traversal of O_e in the drawing of G defines the cyclic ordering π of $\mathrm{mid}(e)$. We always assume that the vertices of every middle set $\mathrm{mid}(e) = V(G_1) \cap V(G_2)$ are enumerated according to π. See Figure 1.7 for an illustration of a sphere-cut decomposition.

1.2.5 Matrix multiplication.

Two $(n \times n)$-matrices can be multiplied using $O(n^\omega)$ algebraic operations, where the naive matrix multiplication shows $\omega \leq 3$. The best upper bound on ω is currently $\omega < 2.376$ [24].

For rectangular matrix multiplication between two $(n \times p)$- and $(p \times n)$-matrices $B = (b_{ij})$ and $C = (c_{ij})$ we differentiate between $p \leq n$ and $p > n$. For the case $p \leq n$ Coppersmith [23] gives an $O(n^{1.85} \cdot p^{0.54})$ time algorithm (under the assumption that $\omega = 2.376$). If $p > n$, we get $O(\frac{p}{n} \cdot n^{2.376} + \frac{p}{n} \cdot n^2)$ by matrix splitting: Split each matrix into $\frac{p}{n}$ many $n \times n$ matrices $B_1, \ldots, B_{\frac{p}{n}}$ and $C_1, \ldots, C_{\frac{p}{n}}$ and multiply each $A_\ell = B_\ell \cdot C_\ell$

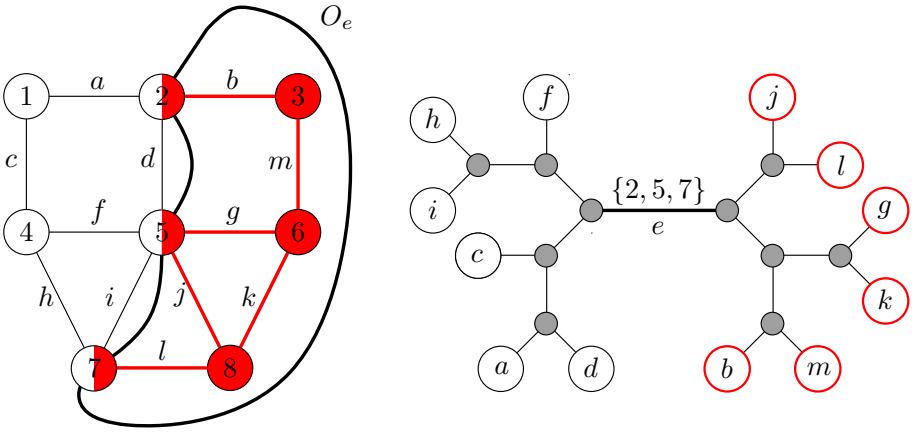


Figure 1.7: The left diagram shows a graph which is separated by a noose two parts– emphasized by the red labeling of the one part. On the right, a sc-decomposition, whose labeled edge corresponds to the noose.

(for all $1 \leq \ell \leq \frac{p}{n}$). Sum up each entry a_{ij}^{ℓ} overall matrices A_ℓ to obtain the solution.

The *distance product* of two $(n \times n)$-matrices B and C, denoted by $B \star C$, is an $(n \times n)$-matrix A such that

$$a_{ij} = \min_{1 \leq k \leq n} \{b_{ik} + c_{kj}\}, 1 \leq i, j \leq n. \tag{1.1}$$

The distance product of two $(n \times n)$-matrices can be computed naively in time $O(n^3)$. Zwick [99] describes a way of using fast matrix multiplication, and fast integer multiplication, to compute distance products of matrices whose elements are taken from the set $\{-m, \ldots, 0, \ldots, m\}$ The running time of the algorithm is $\widetilde{O}(m \cdot n^{\omega})$. For distance product of two $(n \times p)$- and $(p \times n)$-matrices with $p > n$ we get $\widetilde{O}(p \cdot (m \cdot n^{\omega-1}))$ again by matrix splitting: Here we take the minimum of the entries a_{ij}^{ℓ} overall matrices A_ℓ with $1 \leq \ell \leq \frac{p}{n}$.

Another variant is the boolean matrix multiplication. The *boolean matrix multiplication* of two boolean $(n \times n)$-matrices B and C, i.e. with only 0,1-entries, is an boolean $(n \times n)$-matrix A such that

$$a_{ij} = \bigvee_{1 \leq k \leq n} \{b_{ik} \wedge c_{kj}\}, 1 \leq i, j \leq n. \tag{1.2}$$

The fastest algorithm simply uses fast matrix multiplication and sets $a_{ij} = 1$ if $a_{ij} > 0$.

1.2.6 Others

Quotient set. Given a set X and an equivalence relation $\sim$ on X. The *quotient set of* X *by* $\sim$ is the set of all equivalence classes in X given $\sim$ and is denoted by $X/\sim$.

1.3 Unconnected and connected graph problems

In this section, we introduce and classify the problems that are subject of the following chapters. There are two main categories of problems, we are going to investigate. First, we address in Subsection 1.3.1 *vertex-subset problems with local property* **L** , that are graph optimization problems whose solution S is a subset of vertices and where the correctness of S can be verified by "locally" testing for neighborhood relationship satisfying property **L**. This test can be done non-deterministically, i.e., for each vertex one can independently verify **L**. Vertex-subset problems will be our main concern in Chapter 2, where we introduce algorithmic techniques for solving them. Second, we introduce in Subsection 1.3.2 *edge-subset problems with global property* **G**. Here, we search for a subset S of *edges* that satisfy some "global" property, with some neighborhood relations tested in a deterministic way. Note that some *edge-subset problems* can as well be formulated as *vertex-subset problems with global property* **G**. In Chapter 3 we will study how to deal with graph structures of sparse graph classes in order to employ them for designing efficient algorithms for edge-subset problems.

(σ, ϱ)-problems.

A general class of vertex-subset problems are parameterized by two subsets of natural numbers σ and ϱ. A subset of vertices S is a (σ, ϱ)-set if for every $v \in S$ we have $|N(v) \cap S| \in \sigma$ and for every $v \notin S$ we have $|N(v) \cap S| \in \varrho$ [92]. Some well-studied and natural types of (σ, ϱ)-sets are when σ is either all natural numbers, all positive numbers, or $\{0\}$, and with ϱ being either all positive numbers, or $\{1\}$. Some resulting problems are Independent Set ($\sigma = \{0\}$,$\varrho = \mathbb{N}$); Dominating Set ($\sigma = \mathbb{N}$, $\varrho = \mathbb{N}^+$); Perfect Dominating Set ($\sigma = \mathbb{N}$,$\varrho = \{1\}$); Independent Dominating set ($\sigma = \{0\}$,$\varrho = \mathbb{N}^+$); Perfect Code ($\sigma = \{0\}$,$\varrho = \{1\}$); Total Dominating set ($\sigma = \mathbb{N}^+$, $\varrho = \mathbb{N}^+$); Total Perfect Dominating set ($\sigma = \mathbb{N}^+$,$\varrho = \{1\}$). For Perfect Code and Total Perfect Dominating set it is NP-complete even to decide if a graph has any such set, for Independent Dominating set it is NP-complete to find either a smallest or largest such set, while for the remaining three problems it is NP-complete to find a smallest set.

There are many optimization problems that cannot by classified by (σ, ϱ)-sets. Take for example Vertex Cover. For every $v \in S$ we have $\sigma = \mathbb{N}$ but for every $v \notin S$ we have $|N(v) \cap S| = |N(v)|$. In contrast to [92], our definition of vertex-subset problems extends the notion of (σ, ϱ)-sets to properties such as vertex-degree, even-numbered and odd-numbered, and small numbers.

1.3.1 Vertex-subset problems

We define vertex-subset problems as follows:

Definition 1.3.1. *We call a graph optimization problem a* vertex-subset problem with (local) property $\mathbf{L} = (\mathbf{L_1}, \mathbf{L_2})$ *or* generalized (σ, ϱ)-problem *, if for a given graph G*

- *the solution is a subset S of $V(G)$,*
- *for all $v \in S$, $N(v)$ has property $\mathbf{L_1}$,*
- *for all $v \notin S$, $N(v)$ has property $\mathbf{L_2}$.*

Every solution S has an integer value $\mathcal{V}(S)$. If S does not satisfy the previous constraints, we set $\mathcal{V}(S)$ to $+\infty$ for minimization problems and to $-\infty$ for maximization problems. The goal is to find an optimal solution $\mathcal{V} = \min_{S \subseteq V(G)} \mathcal{V}(S)$ for a minimization problem and $\mathcal{V} = \max_{S \subseteq V(G)} \mathcal{V}(S)$ for a maximization problem.

Prominent examples of vertex-subset problems are Independent Set, Vertex Cover, and Dominating Set.

Independent Set: Find $\mathcal{V} = \max_{S \subseteq V(G)} \mathcal{V}(S)$, where

- $\mathcal{V}(S)$ is the cardinality of S;
- $\mathbf{L_1}$: no vertex is in S;
- $\mathbf{L_2}$: arbitrary, i.e., vertices are in $S \cup (V(G) \setminus S)$.

Vertex Cover: Find $\mathcal{V} = \min_{S \subseteq V(G)} \mathcal{V}(S)$, where

- $\mathcal{V}(S)$ is the cardinality of S;
- $\mathbf{L_1}$: arbitrary;
- $\mathbf{L_2}$: all vertices are in S.

Dominating Set: Find $\mathcal{V} = \min_{S \subseteq V(G)} \mathcal{V}(S)$, where

- $\mathcal{V}(S)$ is the cardinality of S;
- $\mathbf{L_1}$: arbitrary;
- $\mathbf{L_2}$: at least one vertex is in S.

In Chapter 2, we will see how these problem are solved optimally for graphs of bounded treewidth by computing the solutions with dynamic programming.

1.3.2 Edge-subset problems

Above vertex-subset problems have another property in common. Given a subset S, one can verify non-deterministically for every vertex separately if S is a legal solution to the problem. That is, we have a "check-operator" that for every vertex v and its neighborhood $N(v)$ independently verifies if property $\mathbf{L}$ is violated or not. If for every v we get an affirmative answer, then the overall solution is legal.

We now consider a class of problems whose solution is an edge subset $E \subseteq E(G)$ and which (usually) has an additional "global" property of solution E and thus gives the entire E as an input to the "check-operator". As global property we consider here, "connected" and "acyclic".

Definition 1.3.2. *We call a graph optimization problem an* edge-subset problem with (global) property $\mathbf{G} = (\mathbf{G_1}, \mathbf{G_2}, \mathbf{G_3})$*, if for a given graph* G

- *the solution is a subset* E *of* $E(G)$*,*
- *for all* $e = \{u, v\} \in E$*,* $\{u, v\}$ *has property* $\mathbf{G_1}$*,*
- *for all* $e = \{u, v\} \notin E$*,* $\{u, v\}$ *has property* $\mathbf{G_2}$*,*
- E *has property* $\mathbf{G_3}$*.*

Every solution E *has integer value* $\mathcal{V}(E)$ *and the goal is to find an optimal solution* $\mathcal{V} = \min_{E \subseteq E(G)} \mathcal{V}(E)$ *for a minimization problem and* $\mathcal{V} = \max_{E \subseteq E(G)} \mathcal{V}(E)$ *for a maximization problem. We also call such a problem* connected problem*.*

We give some of the most important examples:

Longest Cycle: Find $\mathcal{V} = \max_{E \subseteq E(G)} \mathcal{V}(E)$, where

- $\mathcal{V}(E)$ is the cardinality of E;
- $\mathbf{G_1}$: each vertex has two incident edges in E;
- $\mathbf{G_2}$: each vertex has zero or two incident edges in E;
- $\mathbf{G_3}$: connected.

Hamiltonian Cycle: Find a $E \subseteq E(G)$, where

- $\mathcal{V}(E)$ is the cardinality of E and $\mathcal{V}(E) = |V(G)|$;
- $\mathbf{G_1}$: each vertex has two incident edges in E;
- $\mathbf{G_2}$: each vertex has two incident edges in E;
- $\mathbf{G_3}$: connected.

The problems Longest Path and Hamiltonian Path can be described with a slight modification.

In the Metric graph TSP we are given a weighted graph $G = (V, E)$ with weight function $w\colon E(G) \to N$ and we ask for a shortest closed walk that visits all vertices of G at least once. Equivalently, this is TSP with distance metric the shortest path metric of G. Metric graph TSP can be formulated as the edge-subset problem Minimum spanning Eulerian subgraph.

MINIMUM SPANNING EULERIAN SUBGRAPH: Find $\mathcal{V} = \min_{E \subseteq E(G)} \mathcal{V}(E)$,

- $\mathcal{V}(E)$ is the cardinality of E;
- $\mathbf{G_1}$: every vertex has an even, nonzero number of incident edges in E;
- $\mathbf{G_2}$: every vertex has an even, nonzero number of incident edges in E;
- $\mathbf{G_3}$: connected.

Other edge-subset problems that we consider, are for example STEINER TREE and MINIMUM CYCLE COVER.

STEINER TREE WITH VERTEX-SUBSET X: Find $\mathcal{V} = \min_{E \subseteq E(G)} \mathcal{V}(E)$, where

- $\mathcal{V}(E)$ is the cardinality of E;
- $\mathbf{G_1}$: arbitrary;
- $\mathbf{G_2}$: arbitrary, unless vertex in X, then it has at least one incident edge in E;
- $\mathbf{G_3}$: connected.

MINIMUM CYCLE COVER: Find $\mathcal{V} = \min_{E \subseteq E(G)} \mathcal{V}(E)$, where

- $\mathcal{V}(E)$ is the number of cycles formed by E;
- $\mathbf{G_1}$: each vertex has two incident edges in E;
- $\mathbf{G_2}$: each vertex has zero or two incident edges in E;
- $\mathbf{G_3}$: collection of cycles.

We can also consider *vertex-subset problems with global property* that are *vertex-subset problems* according to Definition 1.3.1 with global properties of Definition 1.3.2. The CONNECTED DOMINATING SET problem on an input graph G asks for a minimum dominating subset of vertices that induce a connected subgraph of G and can be formulate MAXIMUM LEAF TREE. Similar, we obtain MAXIMUM INDUCED FORREST on vertex-subset X from the FEEDBACK VERTEX SET problem with solution $V(G) - X$.

MAXIMUM LEAF TREE: Find $\mathcal{V} = \max_{E \subseteq E(G)} \mathcal{V}(E)$, where

- $\mathcal{V}(E)$ is the number of vertices that are incident to exactly one edge of E;
- $\mathbf{G_1}$: at least one vertex has degree ≥ 2 ($|V(G)| > 2$);
- $\mathbf{G_2}$: every vertex has at least one incident edge in E;
- $\mathbf{G_3}$: connected and acyclic.

MAXIMUM INDUCED FORREST: Find $\mathcal{V} = \max_{E \subseteq E(G)} \mathcal{V}(E)$, where

- $\mathcal{V}(E)$ is the number of trees formed by E;

- $\mathbf{G_1}$: arbitrary;
- $\mathbf{G_2}$: at most one vertex has incident edges in E;
- $\mathbf{G_3}$: acyclic.

1.4 Parameterized problems

A *parameter* P is any function mapping graphs to nonnegative integers. The *parameterized problem associated with* P asks, for some fixed k and a minimization problem (maximization problem), whether $P(G) \leq k$ ($P(G) \geq k$) for a given graph G. We say that a parameter P is *closed under taking of minors/contractions* (or, briefly, *minor/contraction closed*) if for every graph H, $H \preceq G$ / $H \preceq_c G$ implies that $P(H) \leq P(G)$.

Many subexponential parameterized graph algorithms [2, 38, 47, 49, 57, 63] are associated with parameters P on graph classes $\mathcal{G}$ satisfying the following two conditions for some constants α and β:

$\boxed{\Lambda}$	For every graph $G \in \mathcal{G}$, its branchwidth $\mathbf{bw}(G) \leq \alpha \cdot \sqrt{P(G)} + O(1)$
$\boxed{\Omega}$	For every graph $G \in \mathcal{G}$ and given a branch-decomposition (T, μ) of G, the value of $P(G)$ can be computed in $2^{\beta \cdot \mathbf{bw}(T,\mu)} n^{O(1)}$ steps.

Conditions $\boxed{\Lambda}$ and $\boxed{\Omega}$ are essential due to the following generic result.

Theorem 1.4.1. *Let P be a parameter and let $\mathcal{G}$ be a class of graphs such that $\boxed{\Lambda}$ and $\boxed{\Omega}$ hold for some constants α and β respectively. Then, given a branch-decomposition (T, μ) where $\mathbf{bw}(T, \mu) \leq \lambda \cdot \mathbf{bw}(G)$ for a constant λ, the parameterized problem associated with P can be solved in $2^{O(\sqrt{k})} n^{O(1)}$ steps.*

Proof. Given a branch-decomposition (T, μ) as above, one can solve the parameterized problem associated with P as follows. If $\mathbf{bw}(T, \mu) > \lambda \cdot \alpha \cdot \sqrt{k}$, then the answer to the associated parameterized problem with parameter k is "NO" if it is a minimization and "YES" if it is a maximization problem. Else, by $\boxed{\Omega}$, $P(G)$ can be computed in $2^{\lambda \cdot \alpha \cdot \beta \cdot \sqrt{k}} n^{O(1)}$ steps. □

To apply Theorem 1.4.1, we need an algorithm that computes, in time $t(n)$, a branch-decomposition (T, μ) of any n-vertex graph $G \in \mathcal{G}$ such that $\mathbf{bw}(T, \mu) \leq \lambda \cdot \mathbf{bw}(G) + O(1)$. Because of [89], $t(n) = n^{O(1)}$ and $\lambda = 1$ for planar graphs. For H-minor-free graphs (and thus, for all graph classes considered here), $t(n) = f(|H|) \cdot n^{O(1)}$ and $\lambda \leq f(|H|)$ for some function f depending only on the size of H (see [37, 43]).

In this section we discuss how to obtain a general scheme of proving bounds required by $\boxed{\Lambda}$ and to extend parameterized algorithms to more general classes of graphs like graphs of bounded genus and graphs excluding a minor (Section 1.4.1).

In Chapter 2, we investigate the techniques used for such algorithms. We will introduce them on the example of the following two vertex-subset problems, that we define here as parameterized problems.

k-Vertex Cover. A *vertex cover* C of a graph is a set of vertices such that every edge of G has at least one endpoint in C. The k-Vertex Cover problem is to decide, given a graph G and a positive integer k, whether G has a vertex cover of size k. Let us note that vertex cover is closed under taking minors, i.e. if a graph G has a vertex cover of size k, then each of its minors has a vertex cover of size at most k.

k-Dominating set. A *dominating* set D of a graph G is a set of vertices such that every vertex outside D is adjacent to a vertex of D. The k-Dominating Set problem is to decide, given a graph G and a positive integer k, whether G has a dominating set of size k. Let us note that the dominating set is not closed under taking minors. However, it is closed under contraction of edges.

Given a branch-decomposition of G of width $\leq \ell$ both problems k-Vertex Cover and k-Dominating Set can be solved in time $2^{O(\ell)}n^{O(1)}$. For the next problem, no such an algorithm is known.

In Chapter 3 we prove that the running time of many dynamic programming algorithms for edge-subset problems on planar graphs (and more general classes as well) satisfies $\boxed{\Omega}$. We take the following edge-subset problem as an example.

k-Longest path. The k-Longest Path problem is to decide, given a graph G and a positive integer k, whether G contains a path of length k. This problem is closed under the operation of taking minor but the best known algorithm solving this problem on a graph of branchwidth $\leq \ell$ runs in time $2^{O(\ell \log \ell)}n^{O(1)}$.

1.4.1 Bidimensionality

In this section we show how to obtain subexponential parameterized algorithms in the case when condition $\boxed{\Omega}$ holds for general graphs. The main tool for this is Bidimensionality Theory developed in [29, 31, 34, 35, 38]. For a survey on Bidimensionality Theory see [33].

Planar graphs. While the results of this subsection can be extended to wider graph classes, we start from planar graphs where the general ideas are easier to explain. The following theorem is the main ingredient for proving condition $\boxed{\Lambda}$.

Theorem 1.4.2 ([86])**.** *Let $\ell \geq 1$ be an integer. Every planar graph of branchwidth $\geq \ell$ contains an $(\ell/4 \times \ell/4)$-grid as a minor.*

We start with Planar k-Vertex Cover as an example. Let G be a planar graph of branchwidth $\geq \ell$. Observe that given a $(r \times r)$-grid H, the size of a vertex cover in

H is at least $\lfloor r/2 \rfloor \cdot r$ (because of the existence of a matching of size $\lfloor r/2 \rfloor \cdot r$ in H). By Theorem 1.4.2, we have that G contains an $(\ell/4 \times \ell/4)$-grid as a minor. The size of any vertex cover of this grid is at least $\ell^2/32$. As such a grid is a minor of G, property $\boxed{\Lambda}$ holds for $\alpha = 4\sqrt{2}$.

For the Planar k-Dominating Set problem, the arguments used above to prove $\boxed{\Lambda}$ for Planar k-Vertex Cover do not work. Since the problem is not minor-closed, we cannot use Theorem 1.4.2 as above. However, since the parameter is closed under edge contractions, we can use a *partially triangulated* $(r \times r)$*-grid* which is any planar graph obtained from the $(r \times r)$-grid by adding some edges. For every partially triangulated $(r \times r)$-grid H, the size of a dominating set in H is at least $\frac{(r-2)^2}{9}$ (every "inner" vertex of H has a closed neighborhood of at most 9 vertices). Theorem 1.4.2 implies that a planar graph G of branchwidth $\geq \ell$ can be contracted to a partially triangulated $(\ell/4 \times \ell/4)$-grid which yields that Planar k-Dominating Set also satisfies $\boxed{\Lambda}$ for $\alpha = 12$.

These two examples induce the following idea: if the graph parameter is closed under taking minors or contractions, the only thing needed for the proof of $\boxed{\Lambda}$ is to understand how this parameter behaves on a (partially triangulated) grid. This brings us to the following definition.

Definition 1.4.3 ([31])**.** *A parameter P is* minor bidimensional with density δ *if*

1. *P is closed under taking of minors, and*
2. *for the $(r \times r)$-grid R, $P(R) = (\delta r)^2 + o((\delta r)^2)$.*

A parameter P is called contraction bidimensional with density δ *if*

1. *P is closed under contractions,*
2. *for any partially triangulated $(r \times r)$-grid R, $P(R) = (\delta_R r)^2 + o((\delta_R r)^2)$, and*
3. *δ is the smallest δ_R among all partially triangulated $(r \times r)$-grids.*

In either case, P is called bidimensional*. The* density δ *of P is the minimum of the two possible densities (when both definitions are applicable), $0 < \delta \leq 1$.*

Intuitively, a parameter is bidimensional if its value depends on the area of a grid and not on its width or height. By Theorem 1.4.2, we have the following.

Lemma 1.4.4. *If P is a bidimensional parameter with density δ then P satisfies property $\boxed{\Lambda}$ for $\alpha = 4/\delta$, on planar graphs.*

Many parameters are bidimensional. Some of them, like the number of vertices or the number of edges, are not so much interesting from an algorithmic point of view. Of course the already mentioned parameter *vertex cover* (*dominating set*) is minor (contraction) bidimensional (with densities $1/\sqrt{2}$ for *vertex cover* and $1/9$ for *dominating set*). Other examples of bidimensional parameters are *feedback vertex set* with density $\delta \in [1/2, 1/\sqrt{2}]$, *minimum maximal matching* with density $\delta \in [1/\sqrt{8}, 1/\sqrt{2}]$ and *longest path* with density 1.

By Lemma 1.4.4, Theorem 1.4.1 holds for every bidimensional parameter satisfying $\boxed{\Omega}$. Also, Theorem 1.4.1 can be applied not only to bidimensional parameters but to parameters that are bounded by bidimensional parameters. For example, the *clique-transversal number* of a graph G is the minimum number of vertices intersecting every maximal clique of G. This parameter is not contraction-closed because an edge contraction may create a new maximal clique and cause the clique-transversal number to increase. On the other hand, it is easy to see that this graph parameter always exceeds the size of a minimum dominating set which yields $\boxed{\Lambda}$ for this parameter.

Non-planar extensions and limitations. One of the natural approaches of extending Lemma 1.4.4 from planar graphs to more general classes of graphs is via extending of Theorem 1.4.2. To do this we have to treat separately minor closed and contraction closed parameters.

The following extension of Theorem 1.4.2 holds for bounded genus graphs:

Theorem 1.4.5 ([31])**.** *If G is a graph of Euler genus at most* $\mathbf{eg}$ *with branchwidth more than r, then G contains a $(r/4(\mathbf{eg}+1) \times r/4(\mathbf{eg}+1))$-grid as a minor.*

Working analogously to the planar case, Theorem 1.4.5 implies the following.

Lemma 1.4.6. *Let P be a minor bidimensional parameter with density δ. Then for any graph G of Euler genus at most* $\mathbf{eg}$*, property $\boxed{\Lambda}$ holds for $\alpha = 4(\mathbf{eg}+1)/\delta$.*

Next step is to consider graphs excluding a fixed graph H as a minor. The proof extends Theorem 1.4.5 by making (nontrivial) use of the structural characterization of H-minor-free graphs by Robertson and Seymour in [85].

Theorem 1.4.7 ([34])**.** *If G is an H-minor-free graph with branchwidth more than r, then G has the $(\Omega(r) \times \Omega(r))$-grid as a minor (the hidden constants in the Ω notation depend only on the size of H).*

As before, Theorem 1.4.5 implies property $\boxed{\Lambda}$ for all minor bidimensional parameters for some α depending only on the excluded minor H.

For contraction-closed parameters, the landscape is different. In fact, each possible extension of Lemma 1.4.6, requires a stronger version of bidimensionality. For this, we can use the notion of a (r,q)-*gridoid* that is obtained from a partially triangulated $(r \times r)$-grid by adding at most q edges. (Note that every (r,q)-gridoid has genus $\leq q$.) The following extends Theorem 1.4.2 for graphs of bounded genus.

Theorem 1.4.8 ([31])**.** *If a graph G of Euler genus at most* $\mathbf{eg}$ *excludes all $(k-12\mathbf{eg}, \mathbf{eg})$-gridoids as contractions, for some $k \geq 12\mathbf{eg}$, then G has branchwidth at most $4k(\mathbf{eg}+1)$.*

A parameter is *genus-contraction bidimensional* if a) it is contraction closed and b) its value on every $(r, O(1))$-gridoid is $\Omega(r^2)$ (here the hidden constants in the big-O and the big-Ω notations depend only on the Euler genus). Then Theorem 1.4.8 implies property $\boxed{\Lambda}$ for all genus-contraction bidimensional parameters for some constant that depends only on the Euler genus.

An *apex graph* is a graph obtained from a planar graph G by adding a vertex and making it adjacent to some vertices of G. A graph class is *apex-minor-free* if it does not contain a graph with some fixed apex graph as a minor. An (r, s)-*augmented grid* is an $(r \times r)$-grid with some additional edges such that each vertex is attached to at most s vertices that in the original grid had degree 4. We say that a contraction closed parameter P is *apex-contraction bidimensional* if a) it is closed under taking of contractions and b) its value on every $(r, O(1))$-augmented grid is $\Omega(r^2)$ (here the hidden constants in the big-O and the big-Ω notations depend only on the excluded apex graph). According to [29] and [34], every apex-contraction bidimensional parameter satisfies property $\boxed{\Lambda}$ for some constant that depends only on the excluded apex graph.

1.4.2 Exact algorithms

We also study problems where we ask for an optimal solution. A problem which does not satisfy property $\boxed{\Lambda}$ is Maximum Independent Set. We can also ask for an optimal solution of Minimum Vertex Cover, Minimum Dominating Set, and Longest Path. Another variant of the latter problem is the Hamiltonian Path, where one asks if there exists a path of length n.

We are interested in finding subexponential exact algorithms for those problems. Therefor, we study the graph classes $\mathcal{G}$ for which we have the following:

$\boxed{\Xi}$	For every graph $G \in \mathcal{G}$, its branchwidth $\mathbf{bw}(G) \leq \gamma \cdot \sqrt{n} + O(1)$
$\boxed{\Omega}$	For every graph $G \in \mathcal{G}$ and given a branch-decomposition (T, μ) of G, the solution can be computed in $2^{\beta \cdot \mathbf{bw}(T,\mu)} n^{O(1)}$ steps.

For the class of planar graphs Property $\boxed{\Xi}$ holds because of the following.

Proposition 1.4.9 ([50]). *For any planar graph G, $\mathbf{bw}(G) \leq \sqrt{4.5n} \leq 2.122\sqrt{n}$.*

Similar results, we obtain for the bounded-genus graph class and the H-minor-free graph class:

We obtain the following result on the branchwidth of 2-cell-embedded graphs on the surface Σ:

Lemma 1.4.10 ([47]). *Given a Σ-embedded graph G where $\Sigma \neq \mathbb{S}_0$, then $\mathbf{bw}(G) \leq (\sqrt{4.5} + 2 \cdot \sqrt{2 \cdot \mathbf{eg}(\Sigma)})\sqrt{n}$.*

We have the following bound on the branchwidth of graphs excluding a fixed sized minor:

Lemma 1.4.11 ([9]). *For any fixed graph H, every H-minor-free graph G has branchwidth $O(\sqrt{n})$.*

That exact vertex subset problems have Property $\boxed{\Omega}$ results from the previous subsection. How to construct such algorithms to be efficient, we will describe in Chapter 2. How edge subset problems, exact and parameterized, satisfy Property $\boxed{\Omega}$ for the three previous graph classes, is subject of Chapter 3.

1.5 Improving dynamic programming on branch- and tree-decompositions

Dynamic programming is a useful tool for fast algorithms solving NP-hard problems. In this chapter, we give an overview of known and new techniques and apply these approaches to vertex-subset problems like k-Dominating Set and k-Vertex cover for giving algorithms on graphs of bounded treewidth and branchwidth.

In Chapter 2 we study several techniques for accelerating the algorithms emerging by the framework of Theorem 1.4.1.

Making algorithms faster. While proving properties $\boxed{\Lambda}$ and $\boxed{\Omega}$, it is natural to ask for the best possible constants α and β, as this directly implies an exponential speed-up of the corresponding algorithms. While, Bidimensionality Theory provides some general estimation of α, in some cases, deep understanding of the parameter behavior can lead to much better constants in $\boxed{\Lambda}$. For example, it holds that for Planar k-Vertex Cover, $\alpha \leq 3$ (see [50]) and for Planar k-Dominating Set, $\alpha \leq 6.364$ (see [49]). (Both bounds are based on the fact that planar graphs with n vertices have branchwidth at most $\sqrt{4.5}\sqrt{n}$, see [50].) Similar results hold also for bounded genus graphs [47].

On the other hand, there are several ways to obtain faster dynamic programming algorithms and to obtain better bounds for β in $\boxed{\Omega}$. On branch-decompositions, a typical approach to compute a solution of size k works as follows (for tree-decompositions similarly):

– Root the branch decomposition (T, μ) of graph G picking any of the vertices of its tree and apply dynamic programming on the middle sets, bottom up, from the leaves toward the root.

– Each middle set $\mathbf{mid}(e)$ of (T, μ) represents the subgraph G_e of G induced by the leaves below. Recall that the vertices of $\mathbf{mid}(e)$ are separators of G.

– In each step of the dynamic programming, all optimal solutions for a subproblem in G_e are computed, subject to all possibilities of how $\mathbf{mid}(e)$ contributes to an overall

solution for G. E.g., for Vertex Cover, there are up to $2^{\mathbf{bw}(T,\mu)}$ subsets of $\mathbf{mid}(e)$ that may constitute a vertex cover of G. Each subset is associated with an optimal solution in G_e with respect to this subset.

– The partial solutions of a middle set are computed using those of the already processed middle sets of the children and stored in an appropriate data structure.

– An optimal solution is computed at the root of T.

Encoding the middle sets in a refined way, may speed up the processing time significantly. Though the same time is needed to scan all solutions assigned to a $\mathbf{mid}(e)$ after introducing vertex states, there are some methods to accelerate the update of the solutions of two middle sets to a parent middle set:

Using the right data structure: storing the solutions in a sorted list or hash-tables compensates the time consuming search for compatible solutions and allows a fast computing of the new solution. E.g., for k-Vertex Cover, the time to process two middle sets is reduced from $O(2^{3\cdot\mathbf{bw}(T,\mu)})$ (for each subset of the parent middle set, all pairs of solutions of the two children are computed) to $O(2^{1.5\cdot\mathbf{bw}(T,\mu)})$. In Section 2.2 matrices are used as a data structure for dynamic programming, that are computed via distance product. The novel technique is based on reducing much of the computation involved to matrix multiplication. The approach is applied to obtain algorithms for various graph problems and allows for example an updating even in time $O(2^{\frac{\omega}{2}\,\mathbf{bw}(T,\mu)})$ for k-Vertex Cover (where ω is the fast matrix multiplication constant, actually $\omega < 2.376$). Since distance product is not known to have a fast matrix multiplication in general, we only consider unweighted and small integer weighted problems with weights of size $m = n^{O(1)}$.

A compact encoding: assign as few as possible vertex states to the vertices and reduce the number of processed solutions. Alber et al. [2], using the so-called "monotonicity technique", show that 3 vertex states are sufficient in order to encode a solution of k-Dominating Set. A similar approach was used in [49] to obtain, for the same problem, a $O(3^{1.5\cdot\mathbf{bw}(T,\mu)})$-step updating process, that is improved in Section 2.2 to $O(2^{2\cdot\mathbf{bw}(T,\mu)})$.

Employing graph structures: With a structural result, presented in Subsection 2.3.1, we will see in Subsection 2.3.2, how one can improve the runtime further for dynamic programming on branch decompositions whose middle sets inherit some structure of the graph. Using such techniques, the update process for Planar k-Dominating Set is done in time $O(3^{\frac{\omega}{2}\,\mathbf{bw}(T,\mu)})$ (see Section 2.2).

In Subsection 2.3.3, we investigate structural properties for tree-decompositions of planar graphs that are used to improve upon the runtime of tree-decomposition based dynamic program- ming approaches for several NP-hard planar graph problems. We give as an example an algorithm for Planar k-Dominating Set of runtime $3^{\mathbf{tw}} \cdot n^{O(1)}$.

Table 1.1 gives all results on algorithms for vertex-subset and edge-subset problems, that are discussed in this thesis. First for general, then for planar graphs in dependency on the parameters treewidth and branchwidth. Last but not least, we give the latest results on parameterized and exact algorithms for some problems.

Table 1.1: Worst-case runtime in the upper part expressed also by treewidth **tw** and branchwidth **bw** of the input graph. The problems marked with '*' are the only one where treewidth may be the better choice for some cut point $\mathbf{tw} \leq \alpha \cdot \mathbf{bw}$ with $\alpha = 1.19$ and 1.05 (compare to Table 2.3 on Page 69). We refer to weighted and unweighted problems, since for applying matrix multiplication we need low weights. Here, unweighted means : weights $M = n^{O(1)}$. The lower part gives a summary of the most important improvements on exact and parameterized algorithms with parameter k. Note that we use the fast matrix multiplication constant $\omega < 2.376$.

	Previous results	New results
Dominating Set (DS)	$O(n2^{\min\{2\,\mathbf{tw},2.38\,\mathbf{bw}\}})$	$O(n2^{2\,\mathbf{bw}})$
Vertex Cover* (VC)	$O(n2^{\mathbf{tw}})$	$O(n2^{\min\{\mathbf{tw},1.19\,\mathbf{bw}\}})$
Independent DS	$O(n2^{\min\{2\,\mathbf{tw},2.38\,\mathbf{bw}\}})$	$O(n2^{2\,\mathbf{bw}})$
Perfect Code*	$O(n2^{\min\{2\,\mathbf{tw},2.58\,\mathbf{bw}\}})$	$O(n2^{\min\{2\,\mathbf{tw},2.09\,\mathbf{bw}\}})$
Perfect DS*	$O(n2^{\min\{2\,\mathbf{tw},2.58\,\mathbf{bw}\}})$	$O(n2^{\min\{2\,\mathbf{tw},2.09\,\mathbf{bw}\}})$
Maximum 2-Packing*	$O(n2^{\min\{2\,\mathbf{tw},2.58\,\mathbf{bw}\}})$	$O(n2^{\min\{2\,\mathbf{tw},2.09\,\mathbf{bw}\}})$
Total DS	$O(n2^{\min\{2.58\,\mathbf{tw},3\,\mathbf{bw}\}})$	$O(n2^{2.58\,\mathbf{bw}})$
Perfect Total DS	$O(n2^{\min\{2.58\,\mathbf{tw},3.16\,\mathbf{bw}\}})$	$O(n2^{2.58\,\mathbf{bw}})$
weighted Planar DS	$O(n2^{\min\{2\,\mathbf{tw},2.38\,\mathbf{bw}\}})$	$O(n2^{1.58\,\mathbf{tw}})$
unweighted Planar DS	$O(n2^{1.89\,\mathbf{bw}})$	$O(n2^{\min\{1.58\,\mathbf{tw},1.89\,\mathbf{bw}\}})$
w Planar Independent DS	$O(n2^{\min\{2\,\mathbf{tw},2.28\,\mathbf{bw}\}})$	$O(n2^{1.58\,\mathbf{tw}})$
uw Planar Independent DS	$O(n2^{1.89\,\mathbf{bw}})$	$O(n2^{\min\{1.58\,\mathbf{tw},1.89\,\mathbf{bw}\}})$
w Planar Total DS	$O(n2^{\min\{2.58\,\mathbf{tw},3\,\mathbf{bw}\}})$	$O(n2^{2\,\mathbf{tw}})$
uw Planar Total DS	$O(n2^{2.38\,\mathbf{bw}})$	$O(n2^{\min\{2\,\mathbf{tw},2.38\,\mathbf{bw}\}})$
w Planar Perfect Total DS	$O(n2^{\min\{2.58\,\mathbf{tw},3.16\,\mathbf{bw}\}})$	$O(n2^{\min\{2.32\,\mathbf{tw},3.16\,\mathbf{bw}\}})$
uw Planar Perfect Total DS	$O(n2^{2.53\,\mathbf{bw}})$	$O(n2^{\min\{2.32\,\mathbf{tw},2.53\,\mathbf{bw}\}})$
w Planar Longest Path	$2^{O(\mathbf{bw}\log\mathbf{bw})} \cdot n^{O(1)}$	$O(n2^{\min\{2.58\,\mathbf{tw},3.37\,\mathbf{bw}\}})$
uw Planar Longest Path	$2^{O(\mathbf{bw}\log\mathbf{bw})} \cdot n^{O(1)}$	$O(n2^{\min\{2.58\,\mathbf{tw},2.75\,\mathbf{bw}\}})$
Planar DS	$O(2^{5.04\sqrt{n}})$ [50]	$O(2^{3.99\sqrt{n}})$
Planar VC	$O(2^{3.18\sqrt{n}})$ [50]	$O(2^{2.52\sqrt{n}})$
w/uw Planar Longest Path	$O(2^{2.29\sqrt{n}\log n})$ [50]	$O(2^{7.2\sqrt{n}})/O(2^{5.83\sqrt{n}})$
w/uw Planar Graph TSP	$2^{O(\sqrt{n}\log n)}$ [72]	$O(2^{9.86\sqrt{n}})/O(2^{8.15\sqrt{n}})$
w/uw Planar connected DS	$2^{O(\sqrt{n}\log n)}$ [72]	$O(2^{9.82\sqrt{n}})/O(2^{8.11\sqrt{n}})$
w/uw Planar Steiner Tree	$2^{O(\sqrt{n}\log n)}$ [72]	$O(2^{8.49\sqrt{n}})/O(2^{7.16\sqrt{n}})$
w/uw Plan Feedback Vertex Set	$2^{O(\sqrt{n}\log n)}$ [72]	$O(2^{9.26\sqrt{n}})/O(2^{7.56\sqrt{n}})$
Parameterized Planar DS	$O(2^{15.13\sqrt{k}}k + n^3)$ [49]	$O(2^{11.98\sqrt{k}}k + n^3)$
Parameterized Planar VC	$O(2^{5.67\sqrt{k}}k + n^3)$ [50]	$O(2^{3.56\sqrt{k}}k + n^3)$
w Param Plan Longest Path	—	$O(2^{13.6\sqrt{k}}k + n^3)$
uw Param Plan Longest Path	—	$O(2^{10.94\sqrt{k}}k + n^3)$

2 Improving dynamic programming techniques

Tree-decompositions or branch-decompositions? The graph parameters treewidth, branchwidth and pathwidth were introduced by Robertson and Seymour as tools in their seminal proof of the Graph Minors Theorem. Recall that the treewidth $\mathbf{tw}(G)$ and branchwidth $\mathbf{bw}(G)$ of a graph G satisfy the relation $\mathbf{bw}(G) \leq \mathbf{tw}(G) + 1 \leq \frac{3}{2}\ \mathbf{bw}(G)$, and thus whenever one of these parameters is bounded by some fixed constant on a class of graphs, then so is the other. Tree-decompositions have traditionally been the choice when solving NP-hard graph problems by dynamic programming to give FPT algorithms when parameterized by treewidth, see e.g. [13, 79] for overviews. Of the various algorithmic templates suggested for this over the years the nice tree-decompositions [66] with binary Join and unary Introduce and Forget operations are preferred for their simplicity and have been widely used both for showing new results, for pedagogical purposes, and in implementations. Tree-decompositions are in fact moving into the computer science curriculum, *e.g.* twenty pages of a new textbook on Algorithm Design [65] is devoted to this topic.

Recently there have been several papers [46, 32, 22, 48, 47] for these FPT algorithms, instead doing the dynamic programming along a branch-decomposition of optimal branchwidth. Dynamic programming along either a branch- or tree-decomposition of a graph both share the property of traversing a tree bottom-up and combining solutions to problems on certain subgraphs that overlap in a bounded-size separator of the original graph. But there are also important differences, *e.g.* the subgraphs mentioned above are for tree-decompositions usually induced by subsets of vertices and for branch-decompositions by non-overlapping sets of edges. A natural question that arises is for which graph classes either parameter, treewidth or branchwidth is the better choice? We will see that in most situations the input graphs contain some graphs where branchwidth is better and others where treewidth is better.

We will first introduce both dynamic programming techniques on the example of k-Vertex Cover. Instead of giving two different dynamic programming approaches for both notions, we introduce in Subsection 2.1.3 a third one the example of k-Dominating Set, into which both decompositions can be formed into without loss of width.

2.1 Dynamic programming: tree-decompositions vs. branch-decompositions

In Subsection 2.1.1 and 2.1.2 we describe how to do dynamic programming on tree-decompositions and branch-decompositions, respectively, using k-Vertex Cover as

an example.

In particular, for a graph G with $|V(G)| = n$ of treewidth **tw** (branchwidth **bw**), we will see how the weighted k-Vertex Cover problem with positive node weights w_v for all $v \in V(G)$ can be solved in time $O(f(\mathbf{tw})) \cdot n^{O(1)}$ ($O(f(\mathbf{bw})) \cdot n^{O(1)}$) where $f(\cdot)$ is an exponential time function only dependent on **tw** (**bw**).

2.1.1 Solving Vertex Cover on tree-decompositions

The algorithm is based on dynamic programming on a rooted tree-decomposition $\mathcal{T} = (T, \mathcal{Z})$ of G. The vertex cover is computed by processing T in post-order from the leaves to the root. For each bag Z_t an optimal vertex cover intersects with some subset U of Z_t. Since Z_t may have size up to **tw**, this may give $2^{\mathbf{tw}}$ possible subsets to consider. The separation property of each bag Z_t ensures that the problems in the different subtrees can be solved independently.

We root T by arbitrarily choosing a node R. Each internal node t of T now has one adjacent node on the path from t to R, called the *parent node*, and some adjacent nodes toward the leaves, called the *children nodes*.

Let T_t be a subtree of T rooted at node t. G_t is the subgraph of G induced by all bags of T_t. For a subset U of $V(G)$ let $w(U)$ denote the total weight of vertices in U. That is, $w(U) = \sum_{u \in U} w_u$. Define a set of subproblems for each subtree T_t. Each set corresponds to a subset $U \subseteq Z_t$ that may represent the intersection of an optimal solution with $V(G_t)$. Thus, for each vertex cover $U \subseteq Z_t$, we denote by $\mathcal{V}_t(U)$ the minimum weight of a vertex cover S in G_t such that $S \cap Z_t = U$, that is $w(S) = \mathcal{V}_t(U)$. We set $\mathcal{V}_t(U) = +\infty$ if U is not a vertex cover since U cannot be part of an optimal solution. There are $2^{|Z_t|}$ possible subproblems associated with each node t of T. Since T has $O(|V(G)|)$ edges, there are in total at most $2^{\mathbf{tw}} \cdot |V(G)|$ subproblems. The minimum weight vertex cover is determined by taking the maximum over all subproblems associated with the root R.

For each node t the information needed to compute $\mathcal{V}_t(U)$ is already computed in the values for the subtrees. For all children nodes $s_1, \ldots, s_\ell$, we simply need to determine the value of the minimum-weight vertex covers S_{s_i} of G_{s_i} $(1 \leq i \leq \ell)$, subject to the constraints that $S_{s_i} \cap Z_t = U \cap Z_{s_i}$.

With vertex covers $U_{s_i} \subseteq Z_{s_i}$ $(1 \leq i \leq \ell)$ that are not necessarily optimal, the value $\mathcal{V}_t(U)$ is given as follows:

$$\mathcal{V}_t(U) = w(U) + \min \quad \{ \quad \sum_{i=1}^{\ell} \mathcal{V}_{s_i}(U_{s_i}) - w(U_{s_i} \cap U) : \quad U_{s_i} \cap Z_t = U \cap Z_{s_i} \}. \tag{2.1}$$

The brute force approach computes for all $2^{|Z_t|}$ sets U associated with t the value $\mathcal{V}_t(U)$ in time $O(\ell \cdot 2^z)$ where $z = \max_i\{|Z_{s_i}|\}$.Hence, the total time spent on node t is $O(4^{\mathbf{tw}} \cdot n)$.

Tables. We will see now how this running time can be improved, namely by the use of a more refined data structure. With a table, one has an object that allows to store all sets $U \subseteq Z_t$ in an ordering such that the time used per node is reduced to $O(2^{\mathbf{tw}} \cdot n)$.

Each node t is assigned a table $Table_t$ that is labeled with the sequence of vertices Z_t. When updating $Table_t$ with $Table_{s_i}$, both tables are sorted by the intersecting vertices $Z_t \cap Z_{s_i}$. For computing $\mathcal{V}_t(U)$, this allows to only process the part of the table $Table_{s_i}$ with $U_{s_i} \cap Z_t = U \cap Z_{s_i}$. Thus both tables get processed only once in total.

For achieving an efficient running time, one uses an adequate encoding of the table entries. First define a coloring $c : V(G) \to \{0, 1\}$: For a node t, each set $U \subseteq Z_t$, if $v \in Z_t \setminus U$ then $c(v) = 0$ else $c(v) = 1$. Then sort $Table_{s_i}$ $(1 \leq i \leq \ell)$ to get entries in an increasing order in order to achieve a fast inquiry.

2.1.2 Solving Vertex Cover on branch-decompositions

On branch-decompositions the algorithm seems to be a bit more circumstantial due to more structure. But as we will see at some point later, this structure is of great help to achieve fast algorithms.

Now we introduce dynamic programming on a rooted branch-decomposition $\langle T, \mu \rangle$ of G. As on tree-decompositions, the vertex cover is computed by processing T in post-order from the leaves to the root. For each middle set $\text{mid}(e)$ an optimal vertex cover intersects with some subset U of $\text{mid}(e)$. Since $\text{mid}(e)$ may have size up to $\mathbf{bw}$, this may give $2^{\mathbf{bw}}$ possible subsets to consider. The separation property of $\text{mid}(e)$ ensures that the problems in the different subtrees can be solved independently.

We root T by arbitrarily choosing an edge e, and subdivide it by inserting a new node s. Let e', e'' be the new edges and set $\text{mid}(e') = \text{mid}(e'') = \text{mid}(e)$. Create a new node *root* r, connect it to s and set $\text{mid}(\{r, s\}) = \varnothing$. Each internal node v of T now has one adjacent edge on the path from v to r, called the *parent edge*, and two adjacent edges toward the leaves, called the *children edges*. To simplify matters, we call them the *left child* and the *right child*.

Let T_e be a subtree of T rooted at edge e. G_e is the subgraph of G induced by all leaves of T_e. For a subset U of $V(G)$ let $w(U)$ denote the total weight of nodes in U. That is, $w(U) = \sum_{u \in U} w_u$. Define a set of subproblems for each subtree T_e. Each set corresponds to a subset $U \subseteq \text{mid}(e)$ that may represent the intersection of an optimal solution with $V(G_e)$. Thus, for each vertex cover $U \subseteq \text{mid}(e)$, we denote by $\mathcal{V}_e(U)$ the minimum weight of a vertex cover S in G_e such that $S \cap \text{mid}(e) = U$, that is $w(S) = \mathcal{V}_e(U)$. We set $\mathcal{V}_e(U) = +\infty$ if U is not a vertex cover since U cannot be part of an optimal solution. There are $2^{|\text{mid}(e)|}$ possible subproblems associated with each edge e of T. Since T has $O(|E(G)|)$ edges, there are in total at most $2^{\mathbf{bw}} \cdot |E(G)|$ subproblems. The minimum weight vertex cover is determined by taking the minimum over all subproblems associated with the root r.

For each edge e the information needed to compute $\mathcal{V}_e(U)$ is already computed in

the values for the subtrees. Since T is ternary, we have that a parent edge e has two children edges f and g. For f and g, we simply need to determine the value of the minimum-weight vertex covers S_f of G_f and S_g of G_g, subject to the constraints that $S_f \cap \mathrm{mid}(e) = U \cap \mathrm{mid}(f)$, $S_g \cap \mathrm{mid}(e) = U \cap \mathrm{mid}(g)$ and $S_f \cap \mathrm{mid}(g) = S_g \cap \mathrm{mid}(f)$.

With vertex covers $U_f \subseteq \mathrm{mid}(f)$ and $U_g \subseteq \mathrm{mid}(g)$ that are not necessarily optimal, the value $\mathcal{V}_e(U)$ is given as follows:

$$\begin{aligned}\mathcal{V}_e(U) = w(U) + \min\{ \quad & \mathcal{V}_f(U_f) - w(U_f \cap U) + \mathcal{V}_g(U_g) - w(U_g \cap U) - w(U_f \cap U_g \setminus U) : \\ & U_f \cap \mathrm{mid}(e) = U \cap \mathrm{mid}(f), \\ & U_g \cap \mathrm{mid}(e) = U \cap \mathrm{mid}(g), \\ & U_f \cap \mathrm{mid}(g) = U_g \cap \mathrm{mid}(f)\}. \end{aligned} \tag{2.2}$$

The brute force approach computes for all $2^{|\,\mathrm{mid}(e)|}$ sets U associated with e the value $\mathcal{V}_e(U)$ in time $O(2^{|\,\mathrm{mid}(f)|} \cdot 2^{|\,\mathrm{mid}(g)|})$. Hence, the total time spent on edge e is $O(8^{\mathbf{bw}})$.

Tables. A more sophisticated approach exploits properties of the middle sets and uses tables as data structure. With a table, one has an object that allows to store all sets $U \subseteq \mathrm{mid}(e)$ in an ordering such that the time used per edge is reduced to $O(2^{1.5\,\mathbf{bw}})$.

By the definition of middle sets, a vertex has to be in at least two of three middle sets of adjacent edges e, f, g. You may simply recall that a vertex has to be in all middle sets along the path between two leaves of T.

For the sake of a refined analysis, we partition the middle sets of parent edge e and left child f and right child g into four sets L, R, F, I as follows:

- *Intersection vertices* $I := \mathrm{mid}(e) \cap \mathrm{mid}(f) \cap \mathrm{mid}(g)$,
- *Forget vertices* $F := \mathrm{mid}(f) \cap \mathrm{mid}(g) \setminus I$,
- *Symmetric difference vertices* $L := \mathrm{mid}(e) \cap \mathrm{mid}(f) \setminus I$ and $R := \mathrm{mid}(e) \cap \mathrm{mid}(g) \setminus I$.

We thus can restate the constraints of Equation (2.2) for the computation of value $\mathcal{V}_e(U)$. Weight $w(U)$ is already contained in $w(U_f \cup U_g)$ since $\mathrm{mid}(e) \subseteq \mathrm{mid}(f) \cup \mathrm{mid}(g)$. Hence, we can change the objective function:

$$\begin{aligned}\mathcal{V}_e(U) = \min\{ \quad & \mathcal{V}_f(U_f) + \mathcal{V}_g(U_g) - w(U_f \cap U_g) : \\ & U_f \cap (I \cup L) = U \cap (I \cup L), \\ & U_g \cap (I \cup R) = U \cap (I \cup R), \\ & U_f \cap (I \cup F) = U_g \cap (I \cup F)\}. \end{aligned} \tag{2.3}$$

Turning to tables, each edge e is assigned a table $Table_e$ that is labeled with the sequence of vertices $\mathrm{mid}(e)$. More precisely, the table is labeled with the concatenation of three sequences out of $\{L, R, I, F\}$. Define the concatenation '$\|$' of two sequences λ_1 and λ_2 as $\lambda_1 \| \lambda_2$. Then, concerning parent edge e and left child f and right child g

we obtain the labels: '$I\|L\|R$' for $Table_e$, '$I\|L\|F$' for $Table_f$, and '$I\|R\|F$' for $Table_g$. $Table_f$ contains all sets U_f with value $\mathcal{V}_f(U_f)$ and analogously, $Table_g$ contains all sets U_g with value $\mathcal{V}_g(U_g)$.

For computing $\mathcal{V}_e(U)$ of each of the $2^{|I|+|L|+|R|}$ entries of $Table_e$, we thus only have to consider $2^{|F|}$ sets U_f and U_g subject to the constraints of Equation (2.3). Since $\text{mid}(e) \cup \text{mid}(f) \cup \text{mid}(g) = I \cup L \cup R \cup F$, we have that $|I| + |L| + |R| + |F| \leq 1.5 \cdot \mathbf{bw}$. Thus we spend in total time $O(2^{1.5\,\mathbf{bw}})$ on each edge of T.

2.1.3 Solving Dominating Set on semi-nice tree decompositions

In this subsection we study how a single dynamic programming algorithm will suffice to give a unified method of using both treewidth, branchwidth and pathwidth.

For this purpose we introduce semi-nice tree-decompositions that maintain much of the simplicity of the nice tree-decompositions. However, the vertices of a Join are partitioned into 3 sets D,E and F, and the binary Join operation treat vertices in each set differently in order to improve runtime. Symmetric Difference vertices D are those that appear in only one of the children, Forget vertices F are those for which all their neighbors have already been considered, and Expensive vertices E are the rest (the formal definitions follow later.) We first see how to transform a given branch-decomposition or tree-decomposition into a semi-nice tree-decomposition. We then will see a template for dynamic programming on semi-nice tree-decompositions for vertex-subset problems.

Semi-nice tree-decompositions

In this paragraph we define semi-nice tree-decompositions and give two algorithms transforming a given branch- or tree-decomposition into a semi-nice tree-decomposition. A tree-decomposition $(T, \mathcal{X})$ is *semi-nice* if T is a rooted binary tree with each non-leaf of T being one of the following:

- **Introduce** node X with a single child C and $C \subset X$.
- **Forget** node X with a single child C and $X \subset C$.
- **Join** node X with two children B, C and $X = B \cup C$.

For an Introduce node we call $X \setminus C$ the 'introduced vertices' and for a Forget node $C \setminus X$ the 'forgotten vertices'. It follows by properties of a tree-decomposition that a vertex can be introduced in several nodes but is forgotten in at most one node. One defines nice tree-decompositions [66] identically to semi-nice tree-decompositions with the following differences:
Join node has $X = B = C$,
Introduce has $|X| = |C| + 1$,
and Forget has $|X| = |C| - 1$.

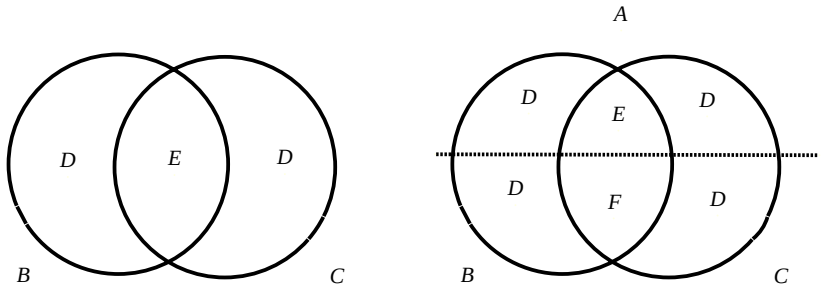


Figure 2.1: Two Venn diagrams illustrating the children B,C of a Join node $X = B \cup C$ and its partition D, E, F. On the right the parent A is a Forget node represented by the part of $B \cup C$ above the dashed line. On the left the parent A is not a Forget node and we then have $B \cup C \subset A$ and $F = \varnothing$. In both cases what we call the New edges go between $B \setminus C$ and $C \setminus B$.

Partition of Join nodes. For a Join node X with children B, C and parent A (the root node being its own parent) we define a partition of $X = B \cup C$ into 3 sets D, I, F:

- **Expensive** $E = A \cap B \cap C$
- **Forgettable** $F = (B \cap C) \setminus A$
- **Symmetric Difference** $D = L \cup R$, where $L = (B \setminus C)$ and $R = (C \setminus B)$

D, E, F is a partition of X by definition. Note that if the parent A of $X = B \cup C$ is an Introduce or Join node then $B \cup C \subset A$ and we get $F = \varnothing$. See Figure 2.1.

Sparse tree-decompositions. The *Forgettable* vertices are useful for any node whose parent is a Forget node, and their definition for an Introduce or leaf node X with parent A is simply $F = X \setminus A$. We say that a neighbor u of a vertex $v \in X$ has been *considered* at node X of T if $u \in X$ or if $u \in X'$ for some descendant node X' of X. Clearly, if X is a Forget node forgetting v then all neighbors of v must have been considered at X. For fast dynamic programming we want sparse semi-nice tree-decompositions where vertices are forgotten as soon as possible.

Definition 2.1.1. *A semi-nice tree-decomposition is* sparse *if whenever a node X containing a vertex $v \in X$ has the property that all neighbors of v have been considered, then the parent of X is a Forget node forgetting v.*

Note that for a Join node with Forget parent A and children B, C of a sparse semi-nice tree-decomposition every vertex in $B \setminus A \cup C$ has a neighbor in $C \setminus A \cup B$ and vice-versa.

Lemma 2.1.2. *Given a tree-decomposition $(T, \mathcal{X})$ of width k of a graph G with n vertices, we can transform it in time $O(k^2 n)$ into a sparse semi-nice tree-decomposition $(T', \mathcal{X}')$ of width k minimizing the E-sets in the partition of each Join node.*

Proof. Choose an arbitrary node as a root and transform T into a binary tree as follows. For each node X having parent A and $d \geq 3$ children $C_1, C_2, ..., C_d$ replace X by a path of $d-1$ nodes, each one with bag X, with one end-node of the path being a child of A, another end-node of the path having child C_d and the remaining $d-1$ children distributed one to each node on the path. Process the tree bottom-up to find for each vertex $v \in V(G)$ a lowest node X_v at which all neighbors of v have been considered, and remove v from any bag that is not a descendant of X_v. Then, for each node A with a single child C such that both $A \setminus C$ and $C \setminus A$ are nonempty, subdivide the edge between A and C by a Forget node $X := C \cap A$. Then, for each node A with two children B, C if $A \setminus (B \cup C)$ is nonempty make a new parent bag $X := B \cup C$ for B and C and make A the parent of X. Finally, for each node A with two children B, C if $B \setminus A$ is nonempty then subdivide the edge between A and B by a Forget node $X := A \cap B$, and likewise for C. The result is a sparse semi-nice tree-decomposition. With a Join node X having parent A and $d \geq 2$ children $C_1, C_2, ..., C_d$ turning into a path, every new Join node X_i has an expensive set $E_i \subseteq A \cap C_i \cap (\bigcup_{k=i-1}^{d} C_k)$. This keeps the resulting Expensive sets E as small as possible when we choose the order of $C_1, C_2, ..., C_d$ arbitrarily. □

Figure 2.2 illustrates the transformation from a branch-decomposition to a semi-nice tree-decomposition described in the following lemma.

Lemma 2.1.3. *Given a branch-decomposition (T, μ) of a graph G with n vertices and m edges we compute a sparse semi-nice tree-decomposition $(T', \mathcal{X})$ with $O(n)$ nodes in time $O(m)$ such that for any bag X of T' we have some $t \in V(T)$ with incident edges e, f, g such that $X \subseteq mid(e) \cup mid(f) \cup mid(g)$ and if X is a Join node with partition D, E, F then $E \subseteq mid(e) \cap mid(f) \cap mid(g)$ and $F \subseteq mid(f) \cap mid(g) \setminus mid(e)$ and $D \subseteq mid(e) \setminus mid(f) \cap mid(g)$.*

Proof. The algorithm has 3 steps:

1) Transform branch-decomposition into a tree-decomposition on the same tree,

2) Transform tree-decomposition into a small tree-decomposition having $O(n)$ nodes,

3) Transform tree-decomposition into a sparse semi-nice tree-decomposition.

Step 1) is well-known (see e.g. [46] for a correctness proof) and proceeds as follows: an inner node t with incident edges e, f, g gets bag $X_t = mid(e) \cup mid(f) \cup mid(g)$, and a leaf node t is assigned a bag containing the two adjacent vertices making the edge $\mu^{-1}(t)$. Root the tree arbitrarily in a leaf. Assume inner node X_t has parent A and children B, C on the end of its incident edges e, f, g, respectively. Note that by construction $X_t = mid(e) \cup mid(f) \cup mid(g)$, and $X_t \cap A = mid(e)$, $X_t \cap B = mid(f)$ and $X_t \cap C = mid(g)$. Assume X_t ends up like this as a Join node after step 3). The partition D, E, F for X_t is then by definition $E = A \cap B \cap C = mid(e) \cap mid(f) \cap mid(g)$, and $F = B \cap C \setminus A = mid(f) \cap mid(g) \setminus mid(e)$, which implies also $D = mid(e) \setminus mid(f) \cap mid(g)$, in agreement with the statement in the lemma. In step 2) we iteratively contract any edge between two nodes with at least one node of degree at most 2 whose bags X, C satisfy $C \subseteq X$ and leave the bag X on the contracted node. In step 3) we apply the

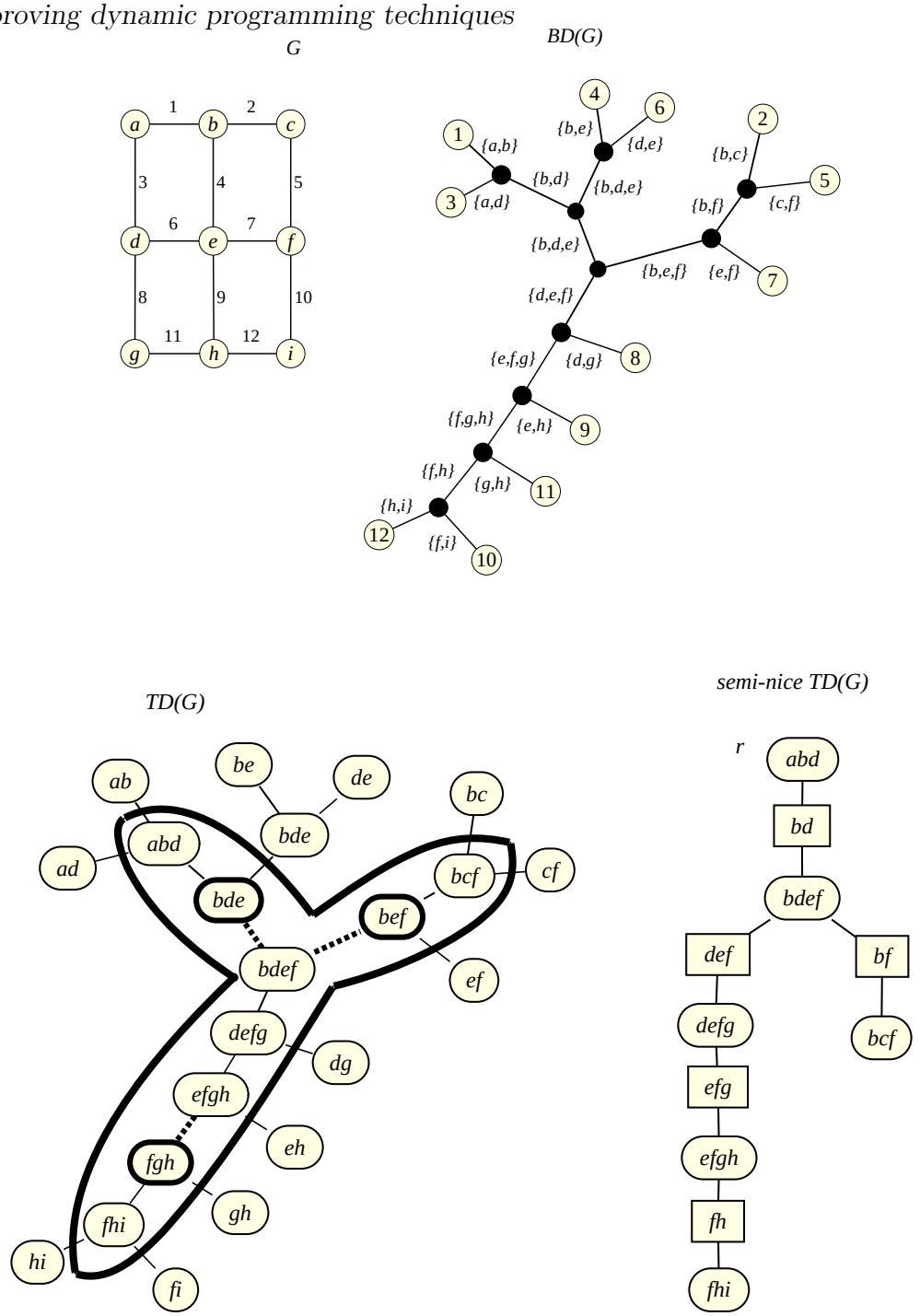


Figure 2.2: On the upper left a 3×3 grid graph G. On the upper right an optimal branch-decomposition with leaves labeled by edges of G as given by μ and the sets $mid(e)$. On the lower left a tree-decomposition formed in the first step of the algorithm of section 2.1.3 with leaf-bags given by μ^{-1} and inner bags given by the union of adjacent $mid(e)$. All nodes outside the bold line are then removed. The edges drawn in a dashed line are contracted and the emphasized bags absorbed by their neighbors. On the lower right the resulting semi-nice tree-decomposition with new nodes emphasized rectangular and arranged below arbitrary root node r.

algorithm from Paragraph 2.1.3 that transforms a tree-decomposition into a sparse semi-nice tree-decomposition $(T', \mathcal{X})$. Steps 2) and 3) did not destroy the property that held for all inner nodes after step 1), and for every node X of T' we can find an original node $t \in V(T)$ with $X \subseteq X_t$. □

Dynamic programming for vertex-subset problems

In this paragraph we give the algorithmic template for doing fast dynamic programming on a semi-nice tree-decomposition $(T, \mathcal{X})$ of a graph G to solve an optimization problem related to vertex subsets on G. The runtime will be given simply as a function of the D, E, F partition of the Join bags, and $X \setminus F, F$ partition of the other bags. In the final paragraph we will then express the runtime by pathwidth, branchwidth or treewidth of the graph. We introduce the template by giving a detailed study of the algorithm for Minimum Dominating Set , and then consider generalizations to various other vertex-subset problems like Perfect Code ,2-Packings and (k, r)-center .

As usual, we compute in a bottom-up manner along the rooted tree T a table of solutions for each node X of T. Let G_X denote the subgraph of G induced by vertices $\{v \in X \text{ or } v \in X' : X' \text{ a descendant of } X \text{ in } T\}$. The table $Table_X$ at X will store solutions to the optimization problem on G_X indexed by the quotient set of all solutions by a problem-specific equivalence relation. The solution to the problem on G is found by an optimization over the table at the root of T. To develop a specific algorithm one must define the tables involved and then show how to Initialize the table at a leaf node of T, how to compute the tables of Introduce, Forget and Join nodes given that their children tables are already computed, and finally how to do the optimization at the root.

Computing Minimum Dominating Set. We use the Minimum Dominating Set problem as an example, whose tables are described by the use of three so-called vertex states:

black: represented by 1, meaning the vertex is in the dominating set.

white: represented by 0, meaning the vertex has a neighbor in G_X that is in the dominating set.

gray: represented by 2, meaning the vertex has a neighbor in G that is in the dominating set. We also call it a *temporary state*.

Theorem 2.1.4. *Given a semi-nice tree-decomposition $(T, \mathcal{X})$ of a graph G on n vertices we can solve in time $O(n(\max\{4^{|E|}3^{|D|+|F|}\} + \max\{|X|3^{|X\setminus F|}2^{|F|}\}))$ the Min Dominating Set Problem on G with maximization over Join nodes of T with partition D, E, F and over Initialization and Introduce nodes with bag X and Forgettable set F, respectively.*

We prove Theorem 2.1.4 in what follows.

Legal vertex subsets. Each index s of $Table_X$ at a node X represents an assignment of states to vertices in the bag X. For index $s : X \to \{1, 0, 2\}$ the vertex subset S of G_X is *legal* for s if:

- $V(G_X) \setminus X = (S \cup N(S)) \setminus X$
- $\{v \in X : s(v) = 1\} = X \cap S$
- $\{v \in X : s(v) = 0\} \subseteq X \cap N(S)$
- $\{v \in X : s(v) = 2\} \subseteq X \setminus S$

$Table_X(s)$ is defined as the cardinality of the smallest S legal for s, or we define $Table_X(s) = \infty$ if no S is legal for s.

Informally, the 4 constraints are that S is a dominating set of $G_X \setminus X$, that vertices with state black are exactly $X \cap S$, and that vertices with state white have a neighbor in S, and that vertices with state gray are simply constrained not to be in S. Since this last constraint is also a constraint on vertices with state white a subset S which is legal for an index s is still legal for an index t where some vertex with state white in s instead has state gray in t. This immediately implies the monotonicity property $Table_X(t) \leq Table_X(s)$ for pairs of indices t and s where $\forall v \in X$ either $t(v) = s(v)$ or $t(v) = 2$ and $s(v) = 0$.

Appropriate data structure. Let us also remark that the $Table_X$ data structure should be an array. To simplify the update operations we should associate integers 0,1,2 with each vertex state so that an index is a 3-ary string of length $|X|$. Moreover, the ordering of vertices in the indices of $Table_X$ should respect the ordering in $Table_C$ for any child node C of X and in case C is the only child of X then all vertices in the larger bag should precede those in the smaller bag. We find this by computing a total order on $V(G)$ respecting the partial order given by the ancestor/descendant relationship of the Forget nodes forgetting vertices $v \in V(G)$.

Processing Forget nodes. The table $Table_X$ at a Forget node X will have $3^{|X|}$ indices, one for each of the possible assignments $s : X \to \{1, 0, 2\}$. We assume a machine model with words of length $3^{|X|}$, to avoid complexity issues related to fast array accesses. Assume Forget node X has child C with $Table_C$ already computed. The correct value for $Table_X(s)$ is the minimum of $\{Table_C(s^+)\}$ over all indices s^+ where $s^+(v) = s(v)$ if $v \in X$ and $s^+(v) \in \{1, 0\}$ otherwise. For this reason we call the state gray a Temporary state. The Forget update operation takes time $O(3^{|X|}2^{|C \setminus X|})$.

Note that the Forget update operation had no need for the indices of the table at the child where a forgotten vertex in $C \setminus X$ had state gray. This observation allows us to save some space and time for the Forgettable vertices of a bag having a Forget parent.

If X is a leaf node with Forgettable vertices F then $Table_X$ has only $3^{|X \setminus F|}2^{|F|}$ indices, in accordance with the above observation, and is computed in a brute-force

manner. This takes time $O(|X|3^{|X\setminus F|}2^{|F|})$, since for each index s we must check if $Table_X(s)$ should be equal to the number of vertices in state black, or if there is a vertex in state white with no neighbor in state black in which case $Table_X(s) = \infty$.

Processing Introduce nodes. If X is an Introduce node with Forgettable vertices F and child C then $Table_X$ has $3^{|X\setminus F|}2^{|F|}$ indices and the correct value at $Table_X(s)$ is:

- ∞ if $Table_C(s) = \infty$ or if $\exists x \in X \setminus C$ with $s(x) = 0$ but no neighbor of x in state black.
- $Table_C(s) + |\{v \in X \setminus C : s(v) = 1\}|$ otherwise

The Introduce update operation thus takes time $O(|X|3^{|X\setminus F|}2^{|F|})$.

Processing Join nodes. The correct values for $Table_X$ at a Join node X with partition D, E, F and children B, C are computed in four steps, where the last three steps account for new adjacency's that have not been considered in any child table (we call these 'new edges'):

1 $\forall s : Table_X(s) = \min\{Table_B(s_b) + Table_C(s_c) - |B \cap C \cap \{v : s(v) = 1\}|\}$ over (s_b, s_c) such that triple (s, s_b, s_c) is necessary (see below).

2 $New = \{uv \in E(G) : u \in B \setminus C \wedge v \in C \setminus B\}$

3 $\forall R \subseteq D : New(R) = \{u \in D \setminus R : \exists v \in R \wedge uv \in New\}$

4 $\forall s : Table_X(s) = Table_X(s')$ where $s'(v) = 0$ if $v \in D \wedge s(v) = 0 \wedge v \in New(\{u : s(u) = 1\})$ and otherwise $s'(v) = s(v)$.

We describe and count the necessary triples of indices (s, s_b, s_c) for the Join update using the method of [46], by first considering the number of *necessary vertex state triples* $(s(v), s_b(v), s_c(v))$ such that vertex state $s_b(v)$ and $s_c(v)$ in B and C respectively will yield the vertex state $s(v)$ in X:

- $v \in B \setminus C \subseteq D$: 3 triples (black,black,-), (white,white,-), (gray,gray,-)
- $v \in C \setminus B \subseteq D$: 3 triples (black,-,black), (white,-,white), (gray,-,gray)
- $v \in F$: 3 triples (black,black,black), (white,gray,white), (white,white,gray)
- $v \in E$: 4 triples (black,black,black), (white,gray,white), (white,white,gray), (gray,gray,gray)

Lemma 2.1.5. *The Join update just described for a node X with partition D, E, F is correct and takes time $O(3^{|D|+|F|}4^{|E|})$.*

Proof. For the timing, the number of necessary triples of indices (s, s_b, s_c) is the product of the number of necessary vertex state triples $(s(v), s_b(v), s_c(v))$ for each vertex in D, E and F, in total $3^{|D|+|F|}4^{|E|}$. For a detailed proof we refer to [46]. The first step in the computation of $Table_X$ therefore takes time $O(3^{|D|+|F|}4^{|E|})$. The remaining steps

compute the $New(S)$ sets and update $Table_X$ within the same time bound (to account for the case $|F \cup E| = O(1)$, for each of the $2^{|F|}3^{|D|+|E|}$ indices s we compute in step 4 the index s' using $O(1)$ operations on words of length at most $3^{|X|}$.)

Let the set of new edges be $New = \{uv \in E(G) : u \in B \setminus C \wedge v \in C \setminus B\}$ as in step 2 of the Join. For correctness, assume $Table_B$ and $Table_C$ are correct and consider an index s of $Table_X$. We will first show that in case we had $New = \varnothing$ then step 1 already computes the correct values. Let therefore S be the smallest vertex subset that is legal for the index s in the graph $G_X \setminus New$, i.e. not accounting for the new edges. Let $S_b = S \cap G_B$ and $S_c = S \cap G_C$. Since $G_B \cap G_C = B \cap C$ we have $S_b \cap S_c = S \cap B \cap C$. Note that the subsets S_b and S_c naturally designate indices s_b and s_c in $Table_B$ and $Table_C$ where all vertices in S_b or S_c have state black, while the remaining have state gray in case they have no neighbors in S_b or S_c and have state white otherwise. The triples $s(v), s_b(v), s_c(v)$ thus constructed from S, S_b, S_c are captured by the definition of necessary vertex state triples, except for the possible triple $(s(x) = 0, s_b(x) = 0, s_c(x) = 0)$. We define index s'_b of $Table_B$ by $s'_b(x) = 2$ for vertices x in such a triple just defined and $s'_b(v) = s_b(v)$ for any other vertex. Note that $(s(x) = 0, s'_b(x) = 2, s_c(x) = 0)$ is a necessary triple. We then have that S_b and S_c are the smallest vertex subsets of G_B and G_C that are legal in $G_X \setminus New$ for the resulting indices s'_b and s_c, by the assumption that S was smallest for index s, and by the monotonicity property $Table_B(s'_b) \leq Table_B(s_b)$. Since s, s'_b, s_c is a necessary triple, the first step of the algorithm will set $Table_X(s)$ to the value $Table_B(s'_b) + Table_C(s_c) - |B \cap C \cap \{v : s(v) = 1\}| = |S_b| + |S_c| - |S_b \cap S_c| = |S|$.

The last three steps will account for the edges in New. The only indices s for which the set of legal subsets are not necessarily the same in $G_X \setminus New$ and G_X are those where there exists a new edge ux with $s(u) = 1$ and $s(x) = 0$. For such an index it is possible that a set S is legal in G_X but that the only neighbor x has in S is the new neighbor u so that S is not legal in $G_X \setminus New$. However, S is legal in $G_X \setminus New$ for the index s' where $s'(x) = 2$ for all $x \in D$ with $s(x) = 0$ having a new neighbor u with $s(u) = 1$, and $s'(v) = s(v)$ otherwise. In case S was the smallest legal subset for s in G_X, the last step of the computation of $Table_X$ would therefore correctly set the value of $Table_X(s)$ to $Table_X(s) = Table_X(s') = |S|$. □

Finally, at the root node R of T we compute the smallest dominating set of G by the minimum of $\{Table_R(s) : s(v) \in \{1, 0\} \forall v \in R\}$. This takes time $O(2^R)$.

Correctness of the algorithm follows by induction on the tree-decomposition, in the standard way for such dynamic programming algorithms.

For the timing we have the Join operation usually being the most expensive, although there are graphs, e.g. when pathwidth=treewidth, for which the leaf Initialization or Introduce operations are the most expensive. However, the Forget and Root optimization operations will never be the most expensive.

Lemma 2.1.5 completes the proof of Theorem 2.1.4. In Section 2.4, we give variations for our algorithm for other domination-type problems.

Runtime by branchwidth, treewidth or pathwidth

In this paragraph we assume that we are given a branch-decomposition of width **bw** or a tree-decomposition of width **tw** and first transform these into a semi-nice tree-decomposition by the algorithms of Paragraph 2.1.3. We then run any of the algorithms described in Paragraph 2.1.3 to express the runtime to solve those problems as a function of **bw** or **tw**. This runtime will match or improve the best results achieved by dynamic programming directly on the branch- or tree-decompositions.

Theorem 2.1.6. *We can solve Minimum Dominating set by dynamic programming on a semi-nice tree-decomposition in time:* $O(2^{3\log_4 3\,\mathbf{bw}}n) = O(2^{2.38\ \mathbf{bw}}n)$ *if given a branch-decomposition* (T,μ) *of width* **bw***;* $O(2^{2\ \mathbf{tw}}n)$ *if given a tree-decomposition of width* **tw***;* $O(2^{1.58\ \mathbf{pw}}n)$ *if given a path-decomposition of width* **pw***; and* $O(2^{\min\{1.58\ \mathbf{pw},2\ \mathbf{tw},2.38\ \mathbf{bw}\}})$ *if given all three. For other domination-type problems we get runtimes as in Table 2.3.*

Proof. We argue in detail only for the Minimum Dominating Set problem, as the other problems in Table 2.3 are handled by completely analogous arguments. Given a branch-decomposition (T,μ) of width **bw** we first transform it into a sparse semi-nice tree-decomposition $(T',\mathcal{X})$ by the algorithm of Lemma 2.1.3. We then apply the algorithm of Theorem 2.1.4. Consider a Join node X with partition D,E,F. By Lemma 2.1.3 D,E,F is related to an inner node $t \in V(T)$ with incident edges e,f,g by $E \subseteq mid(e)\cap mid(f)\cap mid(g)$ and $F \subseteq mid(f)\cap mid(g)\setminus mid(e)$ and $D \subseteq mid(e)\setminus mid(f)\cap mid(g)$. From our definition of middle sets it is easy to see that a vertex $v \in mid(e)\cup mid(f)\cup mid(g)$ appears in at least two out of $mid(e)$, $mid(f)$ and $mid(g)$. From this follows the constraint $|mid(e)\cup mid(f)\cup mid(g)| \leq 1.5\ \mathbf{bw}$ which in addition to the constraints $|mid(e)| \leq \mathbf{bw}, |mid(f)| \leq \mathbf{bw}, |mid(g)| \leq \mathbf{bw}$ gives us four constraints altogether. The worst-case runtime of the Join update of Theorem 2.1.4 is found by taking these four constraints as the constraints of a linear program maximizing $3^{|D|+|F|}4^{|E|} \leq 3^{|(mid(e)\cup mid(f)\cup mid(g))\setminus(mid(e)\cap mid(f)\cap mid(g))|}4^{|mid(e)\cap mid(f)\cap mid(g)|}$. The solution is computed by using an ordinary LP-solver and turns out to occur when $E=\varnothing$ and $|D|+|F| = 1.5\,\mathbf{bw}$, which corresponds to $3^{1.5\,\mathbf{bw}} = 2^{3\log_4 3\,\mathbf{bw}}$. Note that for Introduce nodes we get the same worst-case bound.

If given a tree-decomposition of width **tw** we first transform it into a sparse semi-nice tree-decomposition using Lemma 2.1.2. The worst-case occurs for a Join node with $\mathbf{tw}+1$ Expensive vertices, and Theorem 2.1.4 then gives runtime $O(2^{2\ \mathbf{tw}}n)$. Note that when applying the algorithm of Lemma 2.1.2 to a path-decomposition of width **pw** (which of course is also a tree-decomposition of width **pw**) the resulting sparse semi-nice tree-decomposition will not have any Join nodes. The worst-case runtime then occurs for an Introduce or Root-optimization node X with $\mathbf{pw}+1$ vertices and empty Forgettable set, and Theorem 2.1.4 then gives runtime $O(\mathbf{pw}\ 3^{\mathbf{pw}}n) = O(2^{1.58\ \mathbf{pw}}n)$. □

For certain classes of graphs, e.g. grid graphs, pathwidth is indeed the best parameter for dynamic programming, since $\mathbf{pw} = \mathbf{bw}-1 = \mathbf{tw}$. The runtime we get for Minimum Dominating Set as a function of branchwidth **bw** is essentially the same as

that achieved by the algorithm of [46] working directly on the branch-decomposition (except that the runtime there is expressed with multiplicative factor m instead of our n but we have $m = O(n\ \mathbf{bw})$.)

2.2 Dynamic programming and fast matrix multiplication

The idea of applying fast matrix multiplication is basically to use the information stored in the adjacency matrix of a graph in order to fast detect special subgraphs such as shortest paths, small cliques—as in the previous example—or fixed sized induced subgraphs. Uncommonly we do not use the technique on the graph directly. Instead, it facilitates a fast search in the solution space. In the literature, there has been some approaches speeding up linear programming using fast matrix multiplication, e.g. see [94]. For dynamic programming, we exhibit the properties of overlapping subproblems and optimal substructure. We present a novel approach to fast computing these subsets by applying the distance product on the structure of dynamic programming instead of the graph itself.

2.2.1 How fast matrix multiplication improves algorithms

Fast matrix multiplication gives the currently fastest algorithms for some of the most fundamental graph problems. The main algorithmic tool for solving the All Pair Shortest Paths problem for both directed and undirected graphs with small and large integer weights is to iteratively apply the min-plus product on the adjacency matrix of a graph [88],[90],[8],[99]. Next to the min-plus product or distance product, another variation of matrix multiplication; the boolean matrix multiplication; is solved via fast matrix multiplication. Boolean matrix multiplication is used to obtain the fastest algorithm for Recognizing Triangle-Free Graphs [62]. Recently,Vassilevska and Williams [96] applied the distance product to present the first truly sub-cubic algorithm for finding a Maximum Node-Weighted Triangle in directed and undirected graphs.

The first application of fast matrix multiplication to NP-hard problems was given by Williams [97],who reduced the instances of the well-known problems Max-2-SAT and Max-Cut to Max Triangle on an exponential size graphs dependent on some parameter k, arguing that the optimum weight k-clique corresponds to an optimum solution to the original problem instance.

Max Cut & fast matrix multiplication

In [98], Williams reduces the problems Max-2-SAT and Max Cut to the problem of Max Triangle, that is, finding a maximum weighted 3-clique in a large auxiliary graph. In general, finding 3-cliques in an arbitrary graph G with n vertices can be done in time $O(n^\omega)$: Multiply the adjacency matrix A twice with itself, if A^3 has a non-zero

entry on its main-diagonal, then G contains a 3-clique. The key for solving Max Cut is to reduce the problem with the input graph on n vertices to Max Triangle on an exponentially sized graph, i.e., on $O(2^{\frac{n}{3}})$ vertices and $O(2^{\frac{2n}{3}})$ edges. The problem is then solved via distance product on some type of adjacency matrix of the auxiliary graph.

We will now show here a more straight forward way to solve Max Cut that does the trick directly on matrices without making a detour to Max Triangle. Note that both the idea behind it and the runtime are the same than in [98]. We state this alternative technique here for it is the main key for our algorithm.

Max Cut. The Max Cut problem on a graph G is to find a cut of maximum cardinality, that is, a set $X \subset V(G)$ that maximizes the number of edges between X and $V(G) \setminus X$. Max Cut can be solved in runtime $O(2^n)$ by brute force trying all possible subsets of $V(G)$ and storing the maximum number of edges in the cut.

The following lemma is already stated in [98], the new part is the alternative proof:

Lemma 2.2.1. Max Cut *on a graph with n vertices can be solved in time $O(2^{\omega\frac{n}{3}})$ with $\omega < 2.376$ the fast matrix multiplication factor.*

Proof. Goal is to find the subset X with the optimal value $\mathcal{V}(X)$ to our problem instance. We partition $V(G)$ into three (roughly) equal sets V_1, V_2, V_3. We consider all subsets $X_{i,j} \subseteq V_i \cup V_j$ $(1 \leq i < j \leq 3)$ and define $\mathcal{V}(X_{i,j}) := |\{\{u,v\} \in E(G) \mid u \in X_{i,j}, w \in V_i \cup V_j \setminus X_{i,j}\}|$.

We compute the value $\mathcal{V}(X)$ for an optimal solution X as follows:

$$\begin{aligned}\mathcal{V}(X) = \max\{ \quad & \mathcal{V}(X_{1,2}) + \mathcal{V}(X_{2,3}) - |\{\{u,v\} \in E(G) \mid u \in X_{2,3} \cap V_2, w \in V_2 \setminus X_{2,3}\}| + \\ & \mathcal{V}(X_{1,3}) - |\{\{u,v\} \in E(G) \mid u \in X_{1,3} \cap V_i, w \in V_i \setminus X_{1,3}, (i = 1,3)\}| : \\ & X_{1,2} \cap V_1 = X_{1,3} \cap V_1, \\ & X_{1,2} \cap V_2 = X_{2,3} \cap V_2, \\ & X_{1,3} \cap V_3 = X_{2,3} \cap V_3\}. \end{aligned} \tag{2.4}$$

That is, we maximize the value $\mathcal{V}$ over all subsets $X_{1,2}, X_{1,3}, X_{2,3}$ such that they form an optimal solution X. When summing up the values $\mathcal{V}(X_{1,2}) + \mathcal{V}(X_{1,3}) + \mathcal{V}(X_{2,3})$, we have to subtract the edges that are counted more than once.

We now use distance product to obtain the optimal solution to Max Cut. In order to apply Equation (2.4) and not to count the edges several times, we slightly change the values:

- $\mathcal{V}'(X_{1,2}) := \mathcal{V}(X_{1,2})$;
- $\mathcal{V}'(X_{2,3}) := \mathcal{V}(X_{2,3}) - |\{\{u,v\} \in E(G) \mid u \in X_{2,3} \cap V_2, w \in V_2 \setminus X_{2,3}\}|$;
- $\mathcal{V}'(X_{1,3}) := \mathcal{V}(X_{1,3}) - |\{\{u,v\} \in E(G) \mid u \in X_{1,3} \cap V_i, w \in V_i \setminus X_{1,3}, (i = 1,3)\}|$.

We have that

$$\mathcal{V}(X) = \max\{\mathcal{V}'(X_{1,2}) + \mathcal{V}'(X_{2,3}) + \mathcal{V}'(X_{1,3})\}$$

under the constraints of Equation (2.4).

For $1 \leq i < j \leq 3$, we create the $2^{\frac{n}{3}} \times 2^{\frac{n}{3}}$ matrices $M_{i,j}$ with rows the power set of V_i and columns the power set of V_j. Each row r and column t specify entry $m_{rt}^{i,j} = -\mathcal{V}'(X_{r,t})$ where $X_{r,t} = X_r \cup X_t$ for subsets $X_r \subseteq V_i$ and $X_t \subseteq V_j$. Note that we negate the values because we need to turn the problem into a minimization problem in order to apply the distance product. Then, $\mathcal{V}(X)$ is the minimum entry of $M_{1,3} + (M_{1,2} \star M_{2,3})$,i.e.,

$$\mathcal{V}(X) = \min_{1 \leq r,t \leq 2^{\frac{n}{3}}} \{m_{rt}^{1,3} + m_{rt}^{1,2,3}\},$$

where

$$m_{rt}^{1,2,3} := \min_{1 \leq s \leq 2^{\frac{n}{3}}} \{m_{rs}^{1,2} + m_{st}^{2,3}\}.$$

That is, we first apply distance product on two $2^{\frac{n}{3}} \times 2^{\frac{n}{3}}$ matrices and then add the resulting matrix to a third. Thus, the time we need is $O(n \cdot 2^{\omega\frac{n}{3}} + 2^{\frac{2n}{3}})$. □

2.2.2 How distance product improves dynamic programming

We introduce a dynamic programming approach on branch-decompositions. Instead of using tables, it stores the solutions in matrices that are computed via distance product. Since distance product is not known to have a fast matrix multiplication in general, we only consider unweighted and small integer weighted problems with weights of size $M = n^{O(1)}$.

Matrices. We start again with the example of Vertex Cover and use the notions of Subsection 2.1.2. In the remaining section we show how to use matrices instead of tables as data structure for dynamic programming. Then we apply the distance product of two matrices to compute the values $\mathcal{V}(U)$ for a subset $U \subseteq \text{mid}(e)$ of the parent edge e in the branch-decomposition. Recall also from Subsection 2.1.2 the definitions of intersection-, forget-, and symmetric difference vertices I,F, and L, R, respectively. Reformulating the constraints of Equation (2.3) in the computation of $\mathcal{V}_e(U)$, we obtain:

$$\begin{aligned}\mathcal{V}_e(U) = \min\{ \quad & \mathcal{V}_f(U_f) + \mathcal{V}_g(U_g) - w(U_f \cap U_g) : \\ & U \cap I = U_f \cap I = U_g \cap I, \\ & U_f \cap L = U \cap L, \\ & U_g \cap R = U \cap R, \\ & U_f \cap F = U_g \cap F\}. \end{aligned} \tag{2.5}$$

With $U \cap I = U_f \cap I = U_g \cap I$, one may observe that every vertex cover S_e of G_e is determined by the vertex covers S_f and S_g such that all three sets intersect in some subset $U^I \subseteq I$. From the previous subsection, we got the idea to not compute $\mathcal{V}_e(U)$ for every subset U separately but to simultaneously calculate for each subset $U^I \subseteq I$ the values $\mathcal{V}_e(U)$ for all $U \subseteq \text{mid}(e)$ subject to the constraint that $U \cap I = U^I$. For each of these sets U the values $\mathcal{V}_e(U)$ are stored in a matrix A. A row is labeled with a subset $U^L \subseteq L$ and a column with a subset $U^R \subseteq R$. The entry determined by row U^L and column U^R is filled with $\mathcal{V}_e(U)$ for U subject to the constraints $U \cap L = U^L$, $U \cap R = U^R$, and $U \cap I = U^I$.

We will show how matrix A is computed by the distance product of the two matrices B and C assigned to the children edges f and g: For the left child f, a row of matrix B is labeled with $U^L \subseteq L$ and a column with $U^F \subseteq F$ that appoint the entry $\mathcal{V}_f(U_f)$ for U_f subject to the constraints $U_f \cap L = U^L$, $U_f \cap F = U^F$ and $U_f \cap I = U^I$. Analogously we fill the matrix C for the right child with values for all vertex covers U_g with $U_g \cap I = U^I$. Now we label a row with $U^F \subseteq F$ and a column with $U^R \subseteq R$ storing value $\mathcal{V}_g(U_g)$ for U_g subject to the constraints $U_g \cap F = U^F$ and $U_g \cap R = U^R$. Note that entries have value '$+\infty$' if they are determined by two subsets where at least one set is not a vertex cover.

Lemma 2.2.2. *Given a vertex cover $U^I \subseteq I$. For all vertex covers $U \subseteq \text{mid}(e)$, $U_f \subseteq \text{mid}(f)$ and $U_g \subseteq \text{mid}(g)$ subject to the constraint $U \cap I = U_f \cap I = U_g \cap I = U^I$ let the matrices B and C have entries $\mathcal{V}_f(U_f)$ and $\mathcal{V}_g(U_g)$. The entries $\mathcal{V}_e(U)$ of matrix A are computed by the distance product $A = B \star C$.*

Proof. The rows and columns of A, B and C must be ordered that two equal subsets stand at the same position, i.e., U^L must be at the same position in either row of A and B, U^R in either column of A and C, and U^F must be in the same position in the columns of B as in the rows of C. Note that we set all entries with value '∞' to $\sum_{v \in V(G)} w_v + 1$. Now, the only obstacle from applying the distance product (Section 1.2.5 Equation (1.1)) for our needs, is the additional term $w(U_f \cap U_g)$ in Equation (2.3). Since U_f and U_g only intersect in U^I and U^F, we substitute entry $\mathcal{V}_g(U_g)$ in C for $\mathcal{V}_g(U_g) - |U^I| - |U^F|$ and we get a new equation:

$$\begin{aligned} \mathcal{V}_e(U) = \min \quad \{ \quad & \mathcal{V}_f(U_f) + (\mathcal{V}_g(U_g) - |U^I| - |U^F|) : \\ & U \cap I = U_f \cap I = U_g \cap I = U^I, \\ & U_f \cap L = U \cap L = U^L, \\ & U_g \cap R = U \cap R = U^R, \\ & U_f \cap F = U_g \cap F = U^F\}. \end{aligned} \tag{2.6}$$

Since we have for the worst case analysis that $|L| = |R|$ due to symmetry reason, we may assume that $|U^L| = |U^R|$ and thus A is a square matrix. Every value $\mathcal{V}_e(U)$ in matrix A can be calculated by the distance product of matrix B and C, i.e., by taking the minimum over all sums of entries in row U^L in B and column U^R in C. □

Theorem 2.2.3. *Dynamic programming for the* Vertex Cover *problem on weights* $M = n^{O(1)}$ *on graphs of branchwidth* **bw** *takes time* $\widetilde{O}(M \cdot 2^{\frac{\omega}{2} \cdot \mathbf{bw}})$ *with* ω *the exponent of the fastest matrix multiplication.*

Proof. For every U^I we compute the distance product of B and C with absolute integer values less than M. We show that, instead of a $O(2^{|L|+|R|+|F|+|I|})$ running time, dynamic programming takes time $\widetilde{O}(M \cdot 2^{(\omega-1)|L|} \cdot 2^{|F|} \cdot 2^{|I|})$. We need time $O(2^{|I|})$ for considering all subsets $U^I \subseteq I$. Under the assumption that $2^{|F|} \geq 2^{|L|}$ we get the running time for rectangular matrix multiplication: $\widetilde{O}(M \cdot \frac{2^{|F|}}{2^{|L|}} \cdot 2^{\omega|L|})$. If $2^{|F|} < 2^{|L|}$ we simply get $\widetilde{O}(M \cdot 2^{1.85|L|} \cdot 2^{0.54|F|})$ (for $\omega = 2.376$), so basically the same running time behavior. By the definition of the sets L, R, I, F we obtain four constraints:

- $|I| + |L| + |R| \leq \mathbf{bw}$, since $\mathrm{mid}(e) = I \cup L \cup R$,
- $|I| + |L| + |F| \leq \mathbf{bw}$, since $\mathrm{mid}(f) = I \cup L \cup F$,
- $|I| + |R| + |F| \leq \mathbf{bw}$, since $\mathrm{mid}(g) = I \cup R \cup F$, and
- $|I| + |L| + |R| + |F| \leq 1.5 \cdot \mathbf{bw}$, since $\mathrm{mid}(e) \cup \mathrm{mid}(f) \cup \mathrm{mid}(g) = I \cup L \cup R \cup F$.

When we maximize our objective function $\widetilde{O}(M \cdot 2^{(\omega-1)|L|} \cdot 2^{|F|} \cdot 2^{|I|})$ subject to these constraints, we get the claimed running time of $\widetilde{O}(M \cdot 2^{\frac{\omega}{2} \cdot \mathbf{bw}})$. □

A general technique

Now we formulate the dynamic programming approach using distance product in a more general way than in the previous section in order to apply it to the vertex-subset problems we introduced in Subsection 1.3.1. In the literature these problems are often called *vertex-state* problems. That is, we have given an alphabet λ of vertex-states defined by the corresponding problem. E.g., for the considered Vertex Cover we have that the vertices in the graph have two states relating to an vertex cover U: state '1' means "element of U" and state '0' means "not an element of U". We define a coloring $c : V(G) \to \lambda$ and assign for an edge e of the branch-decomposition $\langle T, \mu \rangle$ a color c to each vertex in $\mathrm{mid}(e)$. Given an ordering of $\mathrm{mid}(e)$, a sequence of vertex-states forms a string $S_e \in \lambda^{|\mathrm{mid}(e)|}$.

Encoding solutions as strings. Recall the definition of concatenating two strings S_1 and S_2 as $S_1 \| S_2$. We then define the strings $S_x(\varrho)$ with $\varrho \in \{L, R, F, I\}$ of length $|\varrho|$ as substrings of S_x with $x \in \{e, f, g\}$ with e parent edge, f left child and g right child. We set $S_e = S_e(I) \| S_e(L) \| S_e(R)$, $S_f = S_f(I) \| S_f(L) \| S_f(F)$ and $S_g = S_g(I) \| S_g(F) \| S_g(R)$. We say that S_e is *formed* by the strings S_f and S_g if $S_e(\varrho)$, $S_f(\varrho)$ and $S_g(\varrho)$ suffice some problem dependent constraints for some $\varrho \in \{L, R, F, I\}$. For Vertex Cover we had in the previous section that S_e is formed by the strings S_f and S_g if $S_e(I) = S_f(I) = S_g(I)$, $S_e(L) = S_f(L)$, $S_e(R) = S_g(R)$ and $S_f(F) = S_g(F)$. For problems as Dominating Set it is sufficient to mention that "formed" is differently defined,

as we saw in Subsection 2.1.3. With the common dynamic programming approach of using tables, we get to proceed $c_1^{|L|} \cdot c_1^{|R|} \cdot c_2^{|F|} \cdot c_3^{|I|}$ update operations of polynomial time where c_1, c_2 and c_3 are small problem dependent constants. Actually, we consider $|\lambda|^{|L|} \cdot |\lambda|^{|F|} \cdot |\lambda|^{|I|}$ solutions of G_f and $|\lambda|^{|R|} \cdot |\lambda|^{|F|} \cdot |\lambda|^{|I|}$ solutions of G_g to obtain $|\lambda|^{|L|} \cdot |\lambda|^{|R|} \cdot |\lambda|^{|I|}$ solutions of G_e. In every considered problem, we have $c_1 \equiv |\lambda|$, $c_2, c_3 \leq |\lambda|^2$ and $c_1 \leq c_2, c_3$.

Generating the matrices. We construct the matrices as follows: For the edges f and g we fix a string $S_f(I) \in \lambda^I$ and a string $S_g(I) \in \lambda^I$ such that $S_f(I)$ and $S_g(I)$ form a string $S_e(I) \in \lambda^I$. We compute a matrix A with $c_1^{|L|}$ rows and $c_1^{|R|}$ columns and with entries $\mathcal{V}_e(S_e)$ for all strings S_e that contain $S_e(I)$. That is, we label monotonically increasing both the rows with strings $S_e(L)$ and the columns with strings $S_e(R)$ that determine the entry $\mathcal{V}_e(S_e)$ subject to the constraint $S_e = S_e(I) \| S_e(L) \| S_e(R)$.

How to do matrix multiplication. Using the distance product, we compute matrix A from matrices B and C that are assigned to the child edges f and g, respectively. Matrix B is labeled monotonically increasing row-wise with strings $S_f(L)$ and column-wise with strings $S_f(F)$. That is, B has $c_1^{|L|}$ rows and $c_2^{|F|}$ columns. A column labeled with string $S_f(F)$ is duplicated depending on how often it contributes to forming the strings $S_e \supset S_e(I)$. The entry determined by $S_f(L)$ and $S_f(F)$ consists of the value $\mathcal{V}_f(S_f)$ subject to $S_f = S_f(I) \| S_f(L) \| S_f(F)$.

Analogously, we compute for edge g the matrix C with $c_2^{|F|}$ rows and $c_1^{|R|}$ columns and with entries $\mathcal{V}_g(S_g)$ for all strings S_g that contain $S_g(I)$. We label the columns with strings $S_g(R)$ and rows with strings $S_g(F)$ with duplicates as for matrix B. However, we do not sort the rows by increasing labels. We order the rows such that the strings $S_g(F)$ and $S_f(F)$ match, where $S_g(F)$ is assigned to row k in C and $S_f(F)$ is assigned to column k in B. I.e., for all $S_f(L)$ and $S_g(R)$ we have that $S_f = S_f(I) \| S_f(L) \| S_f(F)$ and $S_g = S_g(I) \| S_g(F) \| S_g(R)$ form $S_e = S_e(I) \| S_e(L) \| S_e(R)$. The entry determined by $S_g(F)$ and $S_g(R)$ consists of the value $\mathcal{V}_g(S_g)$ subject to $S_g = S_g(I) \| S_g(F) \| S_g(R)$ minus an *overlap*. The overlap is the contribution of the vertex-states of the vertices of $S_g(F) \cap F$ and $S_g(I) \cap I$ to $\mathcal{V}_g(S_g)$. That is, the part of the value that is contributed by $S_g(F) \| S_g(R)$ is not counted since it is already counted in $\mathcal{V}_f(S_f)$.

Lemma 2.2.4. *Consider fixed strings $S_e(I)$, $S_f(I)$ and $S_g(I)$ such that there exist solutions $S_e \supset S_e(I)$ formed by some $S_f \supset S_f(I)$ and $S_g \supset S_g(I)$. The values $\mathcal{V}_f(S_f)$ and $\mathcal{V}_g(S_g)$ are stored in matrices B and C, respectively. Then the values $\mathcal{V}_e(S_e)$ of all possible solutions $S_e \supset S_e(I)$ are computed by the distance product of B and C, and are stored in matrix $A = B \star C$.*

Proof. For all pairs of strings $S_f(L)$ and $S_g(R)$ we compute all possible concatenations $S_e(L) \| S_e(R)$. In row i of B representing one string $S_f(L)$, the values of every string S_f are stored with fixed substrings $S_f(L)$ and $S_f(I)$, namely for all possible substrings $S_f(F)$

labeling the columns. Suppose $S_f(L)$ is updated with string $S_g(R)$ labeling column j of C, i.e., $S_f(L)$ and $S_g(R)$ contribute to forming S_e with substrings $S_e(L)$ and $S_e(R)$. The values of every string $S_g \supset S_g(I)\|S_g(R)$ are stored in that column. For solving a minimization problem we look for the minimum overall possible pairings of $S_f(L)\|S_f(F)$ and $S_g(F)\|S_g(R)$. By construction, a column k of $B = (b_{ij})$ is labeled with the string that that matches the string labeling row k of $C = (c_{ij})$. Thus, the value $\mathcal{V}_e(S_e)$ is stored in matrix A at entry a_{ij} where

$$a_{ij} = \min_k\{b_{ik} + c_{kj}\},\ 1 \leq i \leq c_1^{|L|},\ 1 \leq j \leq c_1^{|R|},\ 1 \leq k \leq c_2^{|F|}.$$

Hence A is the distance product of B and C. □

The following theorem refers especially to all the problems enumerated in Table 2.1.

Theorem 2.2.5. *Dynamic programming for solving vertex-state problems on weights M on graphs of branchwidth* **bw** *takes time $O(M \cdot \max\{c_1^{(\omega-1)\cdot\frac{\mathbf{bw}}{2}} c_2^{\frac{\mathbf{bw}}{2}}, c_2^{\mathbf{bw}}, c_3^{\mathbf{bw}}\})$ with ω the exponent of the fastest matrix multiplication and c_1, c_2 and c_3 the number of algebraic update operations for the sets $\{L, R\}$, F and I, respectively.*

Proof. For each update step we compute for all possible pairings of $S_f(I)$ and $S_g(I)$ the distance product of B and C with absolute integer values less than M. That is, instead of a $c_1^{2\cdot|L|}\cdot c_2^{|F|}\cdot c_3^{|I|}$ running time, dynamic programming takes time $O(M\cdot c_1^{(\omega-1)|L|}\cdot c_2^{|F|}\cdot c_3^{|I|})$. Note that for the worst case analysis we have due to symmetry reason that $|L| = |R|$. We need time $c_3^{|I|}$ for computing all possible pairings of $S_f(I)$ and $S_g(I)$. Under the assumption that $c_2^{|F|} \geq c_1^{|L|}$ we get the running time for rectangular matrix multiplication: $O(M \cdot \frac{c_2^{|F|}}{c_1^{|L|}} \cdot c_1^{\omega|L|})$. If $c_2^{|F|} < c_1^{|L|}$ we simply get $(M \cdot c_1^{1.85|L|} \cdot c_2^{0.54|F|})$ (for $\omega = 2.376$), so basically the same running time behavior.

For parent edge e and child edges f and g, a vertex $v \in \mathrm{mid}(e) \cup \mathrm{mid}(f) \cup \mathrm{mid}(g)$ appears in at least two out of $\mathrm{mid}(e)$, $\mathrm{mid}(f)$ and $\mathrm{mid}(g)$. From this follows the constraint $|\mathrm{mid}(e) \cup \mathrm{mid}(f) \cup \mathrm{mid}(g)| \leq 1.5\ \mathbf{bw}$ which in addition to the constraints $|\mathrm{mid}(e)| \leq \mathbf{bw}, |\mathrm{mid}(f)| \leq \mathbf{bw}, |\mathrm{mid}(g)| \leq \mathbf{bw}$ gives us four constraints altogether: $|L|+|R|+|I|+|F| \leq 1.5\ \mathbf{bw}$, $|L| + |R| + |I| \leq \mathbf{bw}$, $|L| + |I| + |F| \leq \mathbf{bw}$, and $|R| + |I| + |F| \leq \mathbf{bw}$. When we maximize our objective function $O(M \cdot c_1^{(\omega-1)|L|} \cdot c_2^{|F|} \cdot c_3^{|I|})$, we get above result in dependency of the values of c_1, c_2 and c_3. □

Application of the technique

In this section, we show how one can apply the technique for several optimization problems such as Dominating Set and its variants in order to obtain fast algorithms.

For Dominating Set we have that $c_1 \equiv c_2 = 3$ and $c_3 = 4$. The former running time was $O(3^{1.5\,\mathbf{bw}}) \cdot n^{O(1)}$. We have $\widetilde{O}(M \cdot \max\{3^{(\omega-1)\cdot\frac{\mathbf{bw}}{2}} 3^{\frac{\mathbf{bw}}{2}}, 3^{\mathbf{bw}}, 4^{\mathbf{bw}}\}) = \widetilde{O}(M \cdot 4^{\mathbf{bw}})$

Table 2.1: Worst-case runtime in the upper part expressed also by treewidth **tw** and branch-width **bw** of the input graph. The problems marked with '*' are the only one where treewidth may be the better choice for some cut point $\mathbf{tw} \leq \alpha \cdot \mathbf{bw}$ with $\alpha = 1.19$ and 1.05 (compare to Table 2.3 on Page 69). Note that we use the fast matrix multiplication constant $\omega < 2.376$.

	Previous results	New results
Dominating Set	$O(n2^{\min\{2\,\mathbf{tw}, 2.38\,\mathbf{bw}\}})$	$O(n2^{2\,\mathbf{bw}})$
Vertex Cover*	$O(n2^{\mathbf{tw}})$	$O(n2^{\min\{\mathbf{tw}, 1.19\,\mathbf{bw}\}})$
Independent Dominating Set	$O(n2^{\min\{2\,\mathbf{tw}, 2.38\,\mathbf{bw}\}})$	$O(n2^{2\,\mathbf{bw}})$
Perfect Code*	$O(n2^{\min\{2\,\mathbf{tw}, 2.58\,\mathbf{bw}\}})$	$O(n2^{\min\{2\,\mathbf{tw}, 2.09\,\mathbf{bw}\}})$
Perfect Dominating Set*	$O(n2^{\min\{2\,\mathbf{tw}, 2.58\,\mathbf{bw}\}})$	$O(n2^{\min\{2\,\mathbf{tw}, 2.09\,\mathbf{bw}\}})$
Maximum 2-Packing*	$O(n2^{\min\{2\,\mathbf{tw}, 2.58\,\mathbf{bw}\}})$	$O(n2^{\min\{2\,\mathbf{tw}, 2.09\,\mathbf{bw}\}})$
Total Dominating Set	$O(n2^{\min\{2.58\,\mathbf{tw}, 3\,\mathbf{bw}\}})$	$O(n2^{2.58\,\mathbf{bw}})$
Perfect Total Dom Set	$O(n2^{\min\{2.58\,\mathbf{tw}, 3.16\,\mathbf{bw}\}})$	$O(n2^{2.58\,\mathbf{bw}})$

for node weights M if we use a matrix multiplication algorithm with $\omega < 2.5$ and thus hide the factor ω.

Table 2.1 compares the results to Table 2.3 in Section 2.4. It illustrates that dynamic programming is almost always better on branch-decompositions when using fast matrix multiplication rather than dynamic programming on tree decompositions.

2.3 Planarity and dynamic programming

For planar embedded graphs, separators have a structure that cuts the surface into two or more pieces onto which the separated subgraphs are embedded on. Miller [73] was the first to describe how to find small simple cyclic separators in planar triangulations. Applying those ideas, one can find small separators whose vertices can be connected by a closed curve in the plane intersecting the graph only in vertices, so-called *Jordan curves* (e.g. see [11]).

Tree-decompositions and branch-decompositions have been historically the choice when solving NP-hard optimization and FPT problems with a dynamic programming approach (see for example [13] for an overview). Although much is known about the combinatorial structure of tree-decompositions (e.g., [14, 95]) and branch-decompositions (e.g., [49]), few results are known relating to the topology of tree-decompositions or branch-decompositions of planar graphs (e.g., [18, 35, 64].

2.3.1 Computing sphere-cut decompositions

The main idea to speed-up algorithms obtained by the branch-decomposition approach is to exploit planarity for the three times: First in the upper bound on the branchwidth of a

graph and second in the polynomial time algorithm for constructing an optimal branch-decomposition. We present here how for the first time planarity is used in dynamic programming on graphs of bounded branchwidth.

Our results are based on deep results of Seymour & Thomas [89] on geometric properties of planar branch decompositions. Loosely speaking, their results imply that for a graph G embedded on a sphere $\mathbb{S}_0$, some branch decompositions can be seen as decompositions of $\mathbb{S}_0$ into discs (or sphere-cuts). We are the first describing such geometric properties of so-called sphere-cut decompositions (see Section 1.2 for definition). Sphere-cut decompositions seem to be an appropriate tool for solving a variety of planar graph problems.

How to compute sphere-cut decompositions.

We first need to introduce a new notion:

A *carving-decomposition* $\langle T, \mu \rangle$ is similar to a branch-decomposition, only with the difference that μ is the bijection between the leaves of the tree and the *vertex set* of the graph. For an edge e of T, the counterpart of the middle set, called the *cut set* $\mathrm{cut}(e)$, contains the edges of the graph with end vertices in the leaves of both subtrees. The counterpart of branchwidth is carvingwidth.

The following theorem provides us with the main technical tool. It follows almost directly from the results of Seymour & Thomas [89] (see also [56]). Since this result is not explicitly mentioned in [89], we provide some explanations below. Recall the definition of a sc-decomposition (T, μ, π) in Section 1.2.

Theorem 2.3.1. *Let G be a connected Σ-plane graph of branchwidth at most ℓ without vertices of degree one. There exists an sc-decomposition of G of width at most ℓ and such a branch-decomposition can be constructed in time $O(n^3)$.*

Proof. Let G be a Σ-plane graph of branchwidth at most ℓ and with minimal vertex degree at least two. Then, $I(G)$ is the simple bipartite graph with vertices $V(G) \cup E(G)$, in which $v \in V(G)$ is adjacent to $e \in E(G)$ if and only if v is an endpoint of e in G. The *medial graph* M_G of G has vertex set $E(G)$, and for every vertex $v \in V(G)$ there is a cycle C_v in M_G with the following properties:

- The cycles C_v of M_G are mutually edge-disjoint and have as union M_G;
- For each $v \in V(G)$, let the neighbors $w_1, \ldots, w_t$ of v in $I(G)$ be enumerated according to the cyclic order of the edges $\{v, w_1\}$, $\ldots$, $\{v, w_t\}$ in the drawing of $I(G)$; then C_v has vertex set $\{w_1, \ldots, w_t\}$ and w_{i-1} is adjacent to w_i $(1 \leq i \leq t)$, where w_0 means w_t.

In a *bond* carving-decomposition of a graph, every cut set is a bond of the graph, i.e., every cut set is a minimal cut. Seymour and Thomas [89, Theorems (5.1) and (7.2)] show that a Σ-plane graph G without vertices of degree one is of branchwidth at most

ℓ if and only if M_G has a bond carving-decomposition of width at most 2ℓ. They also show [89, Algorithm (9.1)] how to construct an optimal bond carving-decompositions of the medial graph M_G in time $O(n^4)$. A refinement of the algorithm in [56] gives running time $O(n^3)$. A bond carving-decomposition $\langle T, \mu \rangle$ of M_G is also a branch-decomposition of G (vertices of M_G are the edges of G) and it can be shown (see the proof of (7.2) in [89]) that for every edge e of T if the cut set $\mathrm{cut}(e)$ in M_G is of size at most 2ℓ, then the middle set $\mathrm{mid}(e)$ in G is of size at most ℓ. It is well known that the edge set of a minimal cut forms a cycle in the dual graph. The dual graph of a medial graph M_G is the *radial graph* R_G. In other words, R_G is a bipartite graph with the bipartition $F(G) \cup V(G)$. A vertex $v \in V(G)$ is adjacent in R_G to a vertex $f \in F(G)$ if and only if the vertex v is incident to the face f in the drawing of G. Therefore, a cycle in R_G forms a noose in G.

To summarize, for every edge e of T, $\mathrm{cut}(e)$ is a minimal cut in M_G, thus $\mathrm{cut}(e)$ forms a cycle in R_G (and a noose O_e in G). Every vertex of M_G is in one of the open discs Δ_1 and Δ_2 bounded by O_e. Since O_e meets G only in vertices, we have that $O_e \cap V(G) = \mathrm{mid}(e)$. Thus for every edge e of T and the two subgraphs G_1 and G_2 of G formed by the leaves of the subtrees of $T \setminus \{e\}$, O_e bounds the two open discs Δ_1 and Δ_2 such that $G_i \subseteq \Delta_i \cup O_e$, $1 \leq i \leq 2$.

Finally, with a given bond carving-decomposition $\langle T, \mu \rangle$ of the medial graph M_G, it is straightforward to construct a cycle in R_G corresponding to $\mathrm{cut}(e)$, $e \in E(T)$, and afterward to compute the ordering π of $\mathrm{mid}(e)$ in time linear in ℓ. □

2.3.2 Planarity and fast matrix multiplication

Root a given sc-decomposition (T, μ, π). A detailed method can be found in Subsection 2.1.2. Recall that for three neighboring edges of T, the edge closest to the root is called parent edge e_P and the two other edges, child edges e_L and e_R.

Let O_L, O_R, and O_P be the nooses corresponding to edges e_L, e_R and e_P, and let Δ_L, Δ_R and Δ_P be the discs bounded by these nooses. Due to the definition of branch-decompositions, every vertex must appear in at least two of the three middle sets. We partition the set $(O_L \cup O_R \cup O_P) \cap V(G)$ into three sets accordingly to Subsection 2.1.2:

- *Intersection vertices* $I := O_L \cap O_R \cap O_P \cap V(G)$,
- *Forget vertices* $F := O_L \cap O_R \cap V(G) \setminus I$,
- *Symmetric difference vertices* $L := O_P \cap O_L \cap V(G) \setminus I$ and $R := O_P \cap O_R \cap V(G) \setminus I$.

See Figure 2.3 for an illustration of these notions. Observe that $|I| \leq 2$, as the disc Δ_P contains the union of the discs Δ_L and Δ_R. This observation will prove to be crucial for improving dynamic programming for planar graph problems.

With the nice property that $|I| \leq 2$ for all middle sets, we achieve better running times for many discussed problems when we restrict them to planar graphs. The following theorem is the counterpart to Theorem 2.2.5 for planarity:

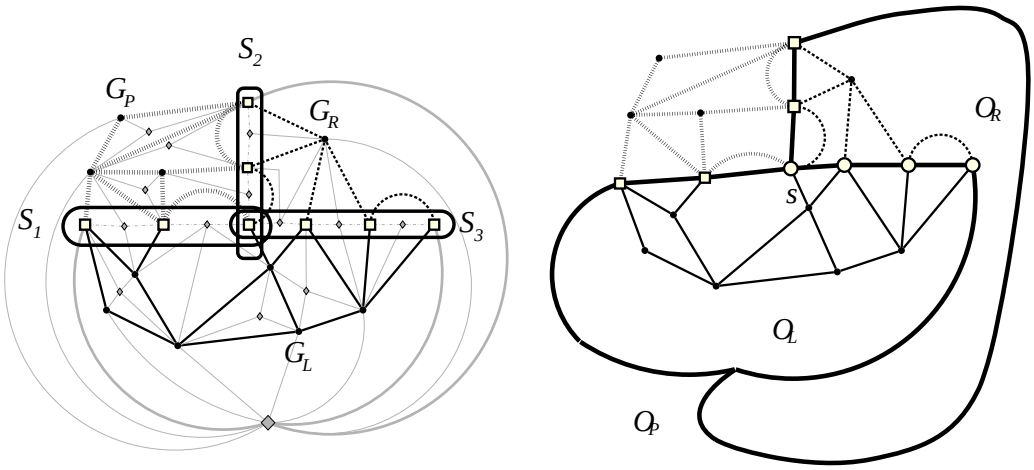


Figure 2.3: On the left we see the same graph G as in the last figure. The gray rhombus and gray edges illustrate the radial graph R_G. G is partitioned by the rectangle vertices of S_1, S_2, S_3 into G_L in drawn-through edges, G_R in dashed edges, and G_P in pointed edges. On the right the three nooses O_L, O_R and O_P are marked. Note that the nooses are induced by S_1, S_2, S_3 and the highlighted gray edges on the left hand. All three nooses here intersect in one intersection vertex s.

Theorem 2.3.2. *Let ω be the exponent of the fastest matrix multiplication and c_1 and c_2 the number of algebraic update operations for the sets $\{L, R\}$ and F, respectively. Then, dynamic programming for solving vertex-subset problems on weights $M = n^{O(1)}$ on planar graphs of branchwidth* **bw** *takes time $\widetilde{O}(M \cdot \max\{c_1^{(\omega-1)\cdot\frac{\mathbf{bw}}{2}} c_2^{\frac{\mathbf{bw}}{2}}, c_2^{\mathbf{bw}}\})$.*

Thus, for Planar Dominating Set with node weights M, the runtime changes from $O(4^{\mathbf{bw}}) \cdot n^{O(1)}$ to $\widetilde{O}(M \cdot 3^{1.188\,\mathbf{bw}}) = \widetilde{O}(M \cdot 3.688^{\mathbf{bw}})$. This runtime is strictly better than the actual runtime of the treewidth based technique of $O(4^{\mathbf{tw}}) \cdot n^{O(1)}$ [2].

2.3.3 Dynamic programming on geometric tree-decompositions

We now study structural properties for tree-decompositions of planar graphs that are used to improve upon the runtime of tree-decomposition based dynamic programming approaches for several NP-hard planar graph problems.

Given any tree-decomposition as an input, we will see how to compute a geometric tree-decomposition that has the same properties as a sc-decomposition. Employing structural results on minimal graph separators for planar graphs, we create in polynomial time a *parallel* tree-decomposition that is assigned by a set of pairwise parallel separators that form pairwise non-crossing Jordan curves in the plane. In a second step, we show

how to obtain a *geometric* tree-decomposition, that has a ternary tree and is assigned Jordan curves that exhaustively decompose the plane into disks (one disk being the infinite disk). In fact, geometric tree-decompositions have all the properties in common with sc-decompositions, that are algorithmically exploited in [49] and in Chapter 3. A similar approach is obtained for 3-connected planar graphs in the proof of self-duality for planar treewidth by Bouchitté et al [18].

Tree-decompositions

Lemma 2.3.3. *[16] Let $\mathcal{T} = (T, \mathcal{Z}), \mathcal{Z} = (Z_t)_{t \in T}$ be a tree-decomposition of G, and let $K \subseteq V(G)$ be a clique in G. Then there exists a node $t \in T$ with $K \subseteq Z_t$.*

As a consequence, we can turn a graph G into another graph H' by saturating the bags of a tree-decomposition, i.e., add an edge in G between any two non-adjacent vertices that appear in a common bag. Automatically, we get that for every clique K in H', there exists a bag Z_t such that $K = Z_t$. Note that the width of the tree-decomposition is not changed by this operation. It is known (e.g. in [95]) that H' is a triangulation of G, actually a so-called *k-tree*. Although there exist triangulations that cannot be computed from G with the elimination game, van Leeuwen [95] describes how to change a tree-decomposition in order to obtain the elimination ordering α and thus $G^+_\alpha = H'$. For finding a minimal triangulation H that is a super-graph of G and a subgraph of G^+_α, known as the *minimal triangulation sandwich* problem, there are efficient $O(nm)$ runtime algorithms. For a nice survey, we refer to [58].

Minimal separators and triangulations

We want to use triangulations for computing tree-decompositions with "nice" separating properties. By Rose et al [87], we have also the following lemma:

Lemma 2.3.4. *Let H be a minimal triangulation of G. Any minimal separator of H is a minimal separator of G.*

Before we give our new tree-decomposition algorithm, we are interested in an additional property of minimal separators. Let $\mathcal{S}_G$ be the set of all minimal separators in G. Let $S_1, S_2 \in \mathcal{S}_G$. We say that S_1 *crosses* S_2, denoted by $S_1 \# S_2$, if there are two connected components $C, D \in G \setminus S_2$, such that S_1 intersects both C and D. Note that $S_1 \# S_2$ implies $S_2 \# S_1$. If S_1 does not cross S_2, we say that S_1 is *parallel* to S_2, denoted by $S_1 || S_2$. Note that "||" is an equivalence relation on a set of pairwise parallel separators.

Theorem 2.3.5. *[77] Let H be a minimal triangulation of G. Then, $\mathcal{S}_H$ is a maximal set of pairwise parallel minimal separators in G.*

Algorithm for a new tree-decomposition

Before we give the whole algorithm, we need some more definitions. For a graph G, let $\mathcal{K}$ be the set of *maximal cliques*, that is, the cliques that have no superset in $V(G)$ that forms a clique in G. Let $\mathcal{K}_v$ be the set of all maximal cliques of G that contain the vertex $v \in V(G)$.For a chordal graph H we define a *clique tree* as a tree $T = (\mathcal{K}, \mathcal{E})$ whose vertex set is the set of maximal cliques in H, and $T[\mathcal{K}_v]$ forms a connected subtree for each vertex $v \in V(H)$. Vice versa, if a graph H has a clique tree, then H is chordal (see [53]). Even though finding all maximal cliques of a graph is NP-hard in general, there exists a linear time modified algorithm of [91], that exploits the property of chordal graphs having at most $|V(H)|$ maximal cliques. By definition, a clique tree of H is also a tree-decomposition of H (where the opposite is not necessarily true).

Due to [12], a clique tree of a chordal graph H is the maximum weight spanning tree of the intersection graph of maximal cliques of H, and we obtain a linear time algorithm computing the clique tree of a graph H. It follows immediately from Lemma 2.3.3 that the treewidth of any chordal graph H equals the size of the largest clique. Let us define an edge (C_i, C_j) in a clique tree T to be equivalent to the set of vertices $C_i \cap C_j$ of the two cliques C_i, C_j in H which correspond to the endpoints of the edge in T. For us, the most interesting property of clique trees is given by [60]:

Theorem 2.3.6. *Given a chordal graph H and some clique tree T of H, a set of vertices S is a minimal separator of H if and only if $S = C_i \cap C_j$ for an edge (C_i, C_j) in T.*

We get our lemma following from Theorem 2.3.5 and Theorem 2.3.6:

Lemma 2.3.7. *Given a clique tree $T = (\mathcal{K}, \mathcal{E})$ of a minimal triangulation H of a graph G. Then, T is a tree-decomposition $\mathcal{T}$ of G, where $\mathbf{tw}(\mathcal{T}) = \mathbf{tw}(H)$, and the set of all edges (C_i, C_j) in T forms a maximal set of pairwise parallel minimal separators in G.*

We call such a tree-decomposition of G *parallel*. We give the algorithm in Figure 3.5.

Algorithm **TransfTD**
Input: Graph G with tree-decomposition $\mathcal{T} = (T, \mathcal{Z}), \mathcal{Z} = (Z_t)_{t \in T}$.
Output: Parallel tree-decomposition $\mathcal{T}'$ of G with $\mathbf{tw}(\mathcal{T}') \leq \mathbf{tw}(\mathcal{T})$.
Triangulation step: Saturate every bag $Z_t, t \in T$ to
obtain the chordal graph H', $E(H') = E(G) \cup F$ with fill edges F.
Minimal triangulation step: Compute a minimal triangulation H of G, $E(H') = E(G) \cup F'$, $F' \subseteq F$.
Clique tree step: Compute the clique tree of H, being simultaneously a tree-decomposition $\mathcal{T}'$ of G.

Figure 2.5: Algorithm **TransfTD**.

The worst case analysis for the runtime of **TransfTD** comes from the Minimal triangulation step, that needs time $O(nm)$ for an input graph G, ($|V(G)| = n$, $|E(G)| = m$).

Plane graphs and minimal separators

In the remainder of the paper, we consider 2-connected plane graphs G. Let $V(J) \subseteq V(G)$ be the set vertices which are intersected by Jordan curve J. The Jordan curve theorem (e.g. see [39]) states that a Jordan curve J on a sphere $\mathbb{S}_0$ divides the rest of $\mathbb{S}_0$ into two connected parts, namely into two open discs Δ_J and $\Delta_{\overline{J}}$, i.e., $\Delta_J \cup \Delta_{\overline{J}} \cup J = \mathbb{S}_0$. Hence, every Jordan curve J is a separator of a plane graph G if both $\Delta_J \cap G$ and $\Delta_{\overline{J}} \cap G$ are nonempty. Two Jordan curves J, J' then divide $\mathbb{S}_0$ into several regions. We define $V^+_{J,J'}$ as the (possibly empty) subset of vertices of $V(J \cap J')$ that are incident to more than two regions. For two Jordan curves J, J', we define $J \Delta J'$ to be the symmetric difference of J and J', and $V(J \Delta J') = V(J \cup J') \setminus V(J \cap J') \cup V^+_{J,J'}$. Bouchitté et al [18] use results of [42] to show the following:

Lemma 2.3.8. *[18] Every minimal separator S of a 2-connected plane graph G forms the vertices of a Jordan curve.*

That is, in any crossing-free embedding of G in $\mathbb{S}_0$, one can find a Jordan curve only intersecting with G in the vertices of S. Note that a minimal separator S is not necessarily forming a unique Jordan curve. If an induced subgraph G' of G (possibly a single edge) has only two vertices u, v in common with S, and u, v are successive vertices of the Jordan curve J, then G' can be drawn on either side of J. This is the only freedom we have to form a Jordan curve in G, since on both sides of J, there is a connected subgraph of G that is adjacent to all vertices of J. We call two Jordan curves J, J' *equivalent* if they share the same vertex set and intersect the vertices in the same order. Two Jordan curves J, J' *cross* if J and J' are not equivalent and there are vertices $v, w \in V(J')$ such that $v \in V(G) \cap \Delta_J$ and $w \in V(G) \cap \Delta_{\overline{J}}$.

Lemma 2.3.9. *Let S_1, S_2 be two minimal separators of a 2-connected plane graph G and each S_i forms a Jordan curve $J_i, i = 1, 2$. If $S_1 || S_2$, then J_1, J_2 are non-crossing. Vice versa, if two Jordan curves J_1, J_2 in G are non-crossing and $\Delta_{J_i} \cap V(G)$ and $\Delta_{\overline{J_i}} \cap V(G), (i = 1, 2)$ all are non-empty, then the vertex sets $S_i = V(J_i), (i = 1, 2)$ are parallel separators.*

We say that two non-crossing Jordan curves J_1, J_2 *touch* if they intersect in a non-empty vertex set. Note that there may exist two edges $e, f \in E(G) \cap \Delta_{J_1}$ such that $e \in E(G) \cap \Delta_{J_2}$ and $f \in E(G) \cap \Delta_{\overline{J_2}}$.

Lemma 2.3.10. *Let two non-crossing Jordan curves J_1, J_2 be formed by two parallel separators S_1, S_2 of a 2-connected plane graph G. If J_1 and J_2 touch, and there exists a Jordan curve $J_3 \subseteq J_1 \Delta J_2$ such that there are vertices of G on both sides of J_3, then the vertices of J_3 form another separator S_3 that is parallel to S_1 and S_2.*

Proof. Let $G_i, \overline{G}_i$ be the subgraphs of G separated by $J_i (i = 1, 2)$. Since the vertex set $V(J_3)$ is a subset of $V(J_1) \cup V(J_2)$ we have that $V(J_3) \cap (V(G_i) \cup V(\overline{G}_i) = \varnothing (i = 1, 2)$. Hence $S_3 = V(J_3)$ is parallel to both, $S_i = V(J_i) (i = 1, 2)$. □

If $J_1 \Delta J_2$ forms exactly one Jordan curve J_3 then we say that J_1 touches J_2 *nicely*. Note that if J_1 and J_2 only touch in one vertex, the vertices of $J_1 \Delta J_2$ may not form any Jordan curve. The following lemma gives a property of "nicely touching"that we need later on.

Lemma 2.3.11. *If in a 2-connected plane graph G, two non-crossing Jordan curves J_1 and J_2 touch nicely, then $|V^+_{J_1,J_2}| = |V(J_1) \cap V(J_2) \cap V(J_1 \Delta J_2)| \leq 2$.*

Jordan curves and geometric tree-decompositions

We now want to turn a parallel tree-decomposition $\mathcal{T}$ into a *geometric* tree-decomposition $\mathcal{T}' = (T, \mathcal{Z}), \mathcal{Z} = (Z_t)_{t \in T}$ where T is a ternary tree and for every two adjacent edges (Z_r, Z_s) and (Z_r, Z_t) in T, the minimal separators $S_1 = Z_r \cap Z_s$ and $S_2 = Z_r \cap Z_t$ form two Jordan curves J_1, J_2 that touch each other nicely. Unfortunately, we cannot arbitrarily connect two Jordan curves J, J' that we obtain from the parallel tree-decomposition $\mathcal{T}$—even if they touch nicely, since the symmetric difference of J, J' may have more vertices than $\mathbf{tw}(\mathcal{T})$. With carefully chosen arguments, one can deduce from [18] that for 3-connected planar graphs parallel tree-decompositions are geometric. However, we give a direct proof that enables us to find geometric tree-decompositions for all planar graphs.

For a vertex set $Z \subseteq V(G)$, we define the subset $\partial Z \subseteq Z$ to be the vertices adjacent in G to some vertices in $V(G) \setminus Z$. Let G be planar embedded, Z connected, and ∂Z form a Jordan curve. We define $\overline{\Delta}_Z$ to be the closed disk, onto which Z is embedded and Δ_Z the open disk with the embedding of Z without the vertices of ∂Z. For a non-leaf tree node X with degree d in a parallel tree-decomposition $\mathcal{T}$, let $Y_1, \dots Y_d$ be its neighbors. Let T_{Y_i} be the subtree including Y_i when removing the edge (Y_i, X) from T. We define $G_{Y_i} \subseteq G$ to be the subgraph induced by the vertices of all bags in T_{Y_i}. For Y_i, choose the Jordan curve J_i formed by the vertex set $\partial Y_i = Y_i \cap X$ to be the Jordan curve that has all vertices of G_{Y_i} on one side and $V(G) \setminus V(G_{Y_i})$ on the other. For each edge e with both endpoints being consecutive vertices of J_i we choose if $e \in E(G_{Y_i})$ or if $e \in E(G) \setminus E(G_{Y_i})$.

We say that a set $\mathcal{J}$ of non-crossing Jordan curves is *connected* if for every partition of $\mathcal{J}$ into two subsets $\mathcal{J}_1, \mathcal{J}_2$, there is at least one Jordan curve of $\mathcal{J}_1$ that touches a Jordan curve of $\mathcal{J}_2$. A set $\mathcal{J}$ of Jordan curves is *k-connected* if for every partition of $\mathcal{J}$ into two connected sets $\mathcal{J}_1, \mathcal{J}_2$, the Jordan curves of $\mathcal{J}_1$ touch the Jordan curves of $\mathcal{J}_2$ in at least k vertices. Note that if two Jordan curves touch nicely then they intersect in at least two vertices.

Lemma 2.3.12. *For every inner node X of a parallel tree-decomposition $\mathcal{T}$ of a 2-connected plane graph, the collection $\mathcal{J}_X$ of pairwise non-crossing Jordan curves formed by ∂X is 2-connected.*

Proof. We first show that $\mathcal{J}_X$ is connected. Assume that $\mathcal{J}_X$ is not connected, that is, there is a partition of $\mathcal{J}_X$ into $\mathcal{J}_1, \mathcal{J}_2$ such that $\mathcal{J}_1$ is connected but no Jordan curve

of $\mathcal{J}_1$ touches any Jordan curve of $\mathcal{J}_2$. We have two cases: first assume that no vertex of the Jordan curves of $\mathcal{J}_1$ is adjacent to any vertex in a Jordan curve of $\mathcal{J}_2$. Each vertex of the Jordan curves of $\mathcal{J}_1$ is adjacent to some vertices in $X_0 := X \setminus \bigcup_{k=1}^{d} Y_k$, for the neighbors $Y_1, \ldots, Y_d$ of X. Hence, there is a Jordan curve J_0 formed exclusively by vertices in X_0 such that $\mathcal{J}_1$ is on one side of J_0 and $\mathcal{J}_2$ on the other. Suppose, there is a pair of vertices u, v where u is a vertex of some Y_i bounded by the Jordan curve $J_i \in \mathcal{J}_1$ and v is a vertex of some Y_j bounded by the Jordan curve $J_j \in \mathcal{J}_2$. By Lemma 2.3.9, J_0 is non-crossing J_i and J_j. Choose J_0 minimal, i.e., no subset of $V(J_0)$ forms a Jordan curve. Thus, $V(J_0) \subseteq X_0$ is a minimal u, v-separator that is parallel to the maximal $\mathcal{S}_G$ set of pairwise parallel minimal separators in G. That is contradicting the maximality of $\mathcal{S}_G$. For the second case assume there are some edges $E_J \subseteq E(X)$ between Jordan curves in $\mathcal{J}_1$ and Jordan curves in $\mathcal{J}_2$. Then there is a closed curve C_J separating $\mathcal{J}_1$ from $\mathcal{J}_2$ touching some (or none) vertices of X_0 and crossing the edges of E_J. Turn C_J into a Jordan curve $J_{1,2}$: for each crossed edge e, move the curve to one endpoint of e, alternately to a vertex of $\mathcal{J}_1$ and a vertex of $\mathcal{J}_2$. Then, $J_{1,2}$ is neither an element of $\mathcal{J}_1$ nor of $\mathcal{J}_2$, and with Lemma 2.3.9 and the same arguments as above, $V(J_{1,2})$ is a minimal separator parallel to $\mathcal{S}_G$ what again is a contradiction to the maximality of $\mathcal{S}_G$.

Now we prove that $\mathcal{J}_X$ is 2-connected. First note that G itself is 2-connected. Thus, if $\mathcal{J}$ is only 1-connected, there must be a path (or edge) in X_0 from some partition $\mathcal{J}_1$ to $\mathcal{J}_2$, if $\mathcal{J}_1$ and $\mathcal{J}_2$ intersect only in one vertex. The proof is very similar to the first case, so we only sketch it. The only difference is that we now assume that there is one vertex w in the intersection of the Jordan curves of $\mathcal{J}_1$ with those of $\mathcal{J}_2$. As in both previous cases, we find a minimal separator S. In the first case, $S \subseteq X_0 \cup \{w\}$ and in the second $S \subseteq X_0 \cup \{w\} \cup V(E_J)$ for the edges E_J with one endpoint in $\mathcal{J}_1$ and the other in $\mathcal{J}_2$. Again, we obtain a contradiction since S is parallel to $\mathcal{S}_G$. □

Lemma 2.3.13. *Every bag X in a parallel tree-decomposition $\mathcal{T}$ can be decomposed into $X_1, \ldots, X_\ell$ such that each vertex set ∂X_i forms a Jordan curve in G and $\bigcup_{i=1}^{\ell} \partial X_i = \partial X$.*

Proof. Let $Y_1, \ldots, Y_d$ be the neighbors of X. By Lemma 2.3.12, ∂X forms a 2-connected set of Jordan curves, each bounding a disk inside which one of the subgraphs G_{Y_j} is embedded onto. If we remove the disks Δ_{Y_j} for all $1 \leq j \leq d$ and the set of Jordan curves $\mathcal{J}_X$ from the sphere, we obtain a collection $\mathcal{D}_X$ of ℓ disjoint open disks each bounded by a Jordan curve of $\mathcal{J}_X$. Note that $\ell \leq \max\{d, |X|\}$. Let Z_i be the subgraph in $X \cap \Delta_i$ for such an open disk $\Delta_i \in \mathcal{D}_X$ for $1 \leq i \leq \ell$. Then each Z_i is either empty or consisting only of edges or subgraphs of G and the closed disk $\overline{\Delta}_i$ is bounded by a Jordan curve J_i formed by a subset of ∂X. We set $X_i = Z_i \cup V(J_i)$ with ∂X_i the vertices of J_i.

□

Lemma 2.3.14. *In a decomposition of the sphere $\mathbb{S}_0$ by a 2-connected collection $\mathcal{J}$ of non-crossing Jordan curves, one can repeatedly find two Jordan curves $J_1, J_2 \in \mathcal{J}$ that touch nicely, and substitute J_1 and J_2 by $J_1 \Delta J_2$ in $\mathcal{J}$.*

Proof. Removing $\mathcal{J}$ from $\mathbb{S}_0$ decomposes $\mathbb{S}_0$ into a collection $\mathcal{D}$ of open discs each bounded by a Jordan curve in $\mathcal{J}$. For each $\Delta_1 \in \mathcal{D}$ bounded by $J_1 \in \mathcal{J}$ there is a "neighboring" disk $\Delta_2 \in \mathcal{D}$ bounded by $J_2 \in \mathcal{J}$ such that the intersection $J_1 \cap J_2$ forms a line of $\mathbb{S}_0$. Then, $J_1 \Delta J_2$ bounds $\Delta_1 \cup \Delta_2$. Replace, J_1, J_2 by J_3 in $\mathcal{J}$ and continue until $|\mathcal{J}| = 1$, that is, we are left with one Jordan curve separating $\mathbb{S}_0$ into two open disks. □

We get that $X_1, \dots X_\ell$ and $G_{Y_1}, \dots, G_{Y_d}$ are embedded inside of closed disks each bounded by a Jordan curve. Thus, the union $\mathcal{D}$ over all these disks together with the Jordan curves $\mathcal{J}_X$ fill the entire sphere $\mathbb{S}_0$ onto which G is embedded. Each subgraph embedded onto $\Delta \cup J$ for a disk $\Delta \in \mathcal{D}$ and a Jordan curve J bounding Δ, forms either a bag X_i or a subgraph G_{Y_j}. Define the collection of bags $\mathcal{Z}^X = \{X_1, \dots X_\ell, Y_1, \dots, Y_d\}$. In Figure 2.6, we give the algorithm **TransfTD II** for creating a geometric tree-decomposition using the idea of Lemma 2.3.10.

Algorithm **TransfTD II**
Input: Graph G with parallel tree-decomposition $\mathcal{T} = (T, \mathcal{Z}), \mathcal{Z} = (Z_t)_{t \in T}$.
Output: Geometric tree-decomposition $\mathcal{T}'$ of G with $\mathbf{tw}(\mathcal{T}') \leq \mathbf{tw}(\mathcal{T})$.

For each inner bag X with neighbors $Y_1, \dots, Y_d$ {
Disconnection step: Replace X by $X_1, \dots X_\ell$ (Lemma 2.3.13). Set $\mathcal{Z}^X = \{X_1, \dots X_\ell, Y_1, \dots, Y_d\}$.
Reconnection step: Until $|\mathcal{Z}^X| = 1$ {
  Find two bags Z_i and Z_j in $\mathcal{Z}^X$ such that Jordan curve $J_i \Delta J_j$
  bounds a disk with $Z_i \cup Z_j$ (Lemma 2.3.14);
  Set $Z_{ij} = (Z_i \Delta Z_j) \cup (Z_i \cap Z_j)$ and connect Z_i and Z_j to Z_{ij};
  In $\mathcal{Z}^X$: substitute Z_i and Z_j by Z_{ij}. }}

Figure 2.6: Algorithm **TransfTD II**.

Since by Lemma 2.3.11, $|V(\partial Z_i \cap \partial Z_j \cap \partial Z_{ij})| \leq 2$, we have that at most two vertices in all three bags are contained in any other bag of $\mathcal{Z}^X$. Note that geometric tree-decompositions have a lot in common with *sphere-cut decompositions*, namely that both decompositions are assigned with vertex sets that form "sphere-cutting" Jordan curves. For our new dynamic programming algorithm, we use much of the structure results obtained in Subsection 2.3.1.

Jordan curves and dynamic programming

The following techniques improve the existing algorithm of Alber et al [2] for weighted Planar Dominating Set. Their algorithm is based on dynamic programming on *nice tree-decompositions* $\mathcal{T}$ and has the running time $4^{\mathbf{tw}(\mathcal{T})} \cdot n^{O(1)}$. We prove the following theorem by giving an algorithm of similar structure to the one in Subsection 2.1.3. Thus, we give here only a sketch of the idea. Namely, to exploit the planar structure of the nicely touching separators to improve upon the runtime.

Theorem 2.3.15. *Given a geometric tree-decomposition $\mathcal{T} = (T, \mathcal{Z}), \mathcal{Z} = (Z_t)_{t\in T}$ of a planar graph G. Weighted* Planar Dominating Set *on G can be solved in time $3^{\mathbf{tw}(\mathcal{T})} \cdot n^{O(1)}$.*

Proof. We root T by arbitrarily choosing a node r as a *root*. Each internal node t of T now has one adjacent node on the path from t to r, called the *parent node*, and two adjacent nodes toward the leaves, called the *children nodes*. To simplify matters, we call them the *left child* and the *right child*.

Let T_t be a subtree of T rooted at node t. G_t is the subgraph of G induced by all bags of T_t. For a subset U of $V(G)$ let $w(U)$ denote the total weight of vertices in U. That is, $w(U) = \sum_{u\in U} w_u$. Define a set of subproblems for each subtree T_t.

Alber et al. [2] introduced the so-called "monotonicity"-property of domination-like problems for their dynamic programming approach that we will use, too. For every node $t \in T$, we use three colors for the vertices of bag Z_t:

black: represented by 1, meaning the vertex is in the dominating set.

white: represented by 0, meaning the vertex has a neighbor in G_t that is in the dominating set.

gray: represented by 2, meaning the vertex has a neighbor in G that is in the dominating set.

For a bag Z_t of cardinality ℓ, we define a *coloring* $c(Z_t)$ to be a mapping of the vertices Z_t to an ℓ-vector over the color-set $\{0, 1, 2\}$ such that each vertex $u \in Z_t$ is assigned a color, i.e., $c(u) \in \{0, 1, 2\}$. We further define the weight $w(c(Z_t))$ to be the minimum weight of the vertices of G_t in the minimum weight dominating set with respect to the coloring $c(Z_t)$. If no such dominating set exists, we set $w(c(Z_t)) = +\infty$. We store all colorings of Z_t, and for two child nodes, we update each two colorings to one of the parent node.

Before we describe the updating process of the bags, let us make the following comments:

We defined the color "gray" according to the monotonicity property: for a vertex u colored gray, we do not have (or store) the information if u is already dominated by a vertex in G_t or if u still has to be dominated in $G \setminus G_t$. Thus, a solution with a vertex v colored white has at least the same the weight as the same solution with v colored gray.

By the definition of bags, for three adjacent nodes r, s, t, the vertices of ∂Z_r have to be in at least on of ∂Z_s and ∂Z_t. The reader may simply recall that the parent bag is formed by the union of the vertices of two nicely touching Jordan curves.

For the sake of a refined analysis, we partition the bags of parent node r and left child s and right child t into four sets L, R, F, I as follows:

- *Intersection* $I := \partial Z_r \cap \partial Z_s \cap \partial Z_t$,
- *Forget* $F := (Z_s \cup Z_t) \setminus \partial Z_r$,

• *Symmetric difference* $L := \partial Z_r \cap \partial Z_s \setminus I$ and $R := \partial Z_r \cap \partial Z_t \setminus I$.

We define F' to be actually those vertices of F that are only in $(\partial Z_s \cup \partial Z_t) \setminus \partial Z_r$. The vertices of $F \setminus F'$ do not exist in Z_r and hence are irrelevant for the continuous update process. We say that a coloring $c(Z_r)$ is *formed* by the colorings $c_1(Z_s)$ and $c_2(Z_t)$ subject to the following rules:

($R1$) For every vertex $u \in L \cup R$: $c(u) = c_1(u)$ and $c(u) = c_2(u)$, respectively.

($R2$) For every vertex $u \in F'$ either $c(u) = c_1(u) = c_2(u) = 1$ or $c(u) = 0 \wedge c_1(u), c_2(u) \in \{0, 2\} \wedge c_1(u) \neq c_2(u)$.

($R3$) For every vertex $u \in I$ $c(u) \in \{1, 2\} \Rightarrow c(u) = c_1(u) = c_2(u)$ and $c(u) = 0 \Rightarrow c_1(u), c_2(u) \in \{0, 2\} \wedge c_1(u) \neq c_2(u)$.

We define U_c to be the vertices $u \in Z_s \cap Z_t$ for which $c(u) = 1$ and update the weights by:

$$w(c(Z_r)) = \min\{w(c_1(Z_s)) + w(c_2(Z_t)) - w(U_c) | c_1, c_2 \text{ forms } c\}$$

.

The number of steps by which $w(c(Z_r))$ is computed for every possible coloring of Z_r is given by the number of ways a color c can be formed by the three rules $(R1), (R2), (R3)$, i.e.,

$$3^{|L|+|R|} \cdot 3^{|F'|} \cdot 4^{|I|}$$

steps.

By Lemma 2.3.11, $|I| \leq 2$ and since $|L|+|R|+|F| \leq \mathbf{tw}(\mathcal{T})$, we need at most $3^{\mathbf{tw}(\mathcal{T})} \cdot n$ steps to compute all weights $w(c(Z_r))$ that are usually stored in a table assigned to bag Z_r.

In [2], the worst case in the runtime for Planar Dominating Set is determined by the number of vertices that are in the intersection of three adjacent bags r, s, t. Using the notion of Subsection 2.1.2 for a geometric tree-decomposition, we partition the vertex sets of three bags Z_r, Z_s, Z_t into sets L, R, F, I, where Z_r is adjacent to Z_s, Z_t. The sets L, R, F represent the vertices that are in exactly two of the bags. Let us consider the *Intersection* set $I := \partial Z_r \cap \partial Z_s \cap \partial Z_t$. By Lemma 2.3.11, $|I| \leq 2$. Thus, I is not any more part of the runtime.

□

Summary of problems and solutions. In Section 2.2, we combined dynamic programming with fast matrix multiplication to get $4^{\mathbf{bw}} \cdot n^{O(1)}$ and for Planar Dominating Set even $3^{\frac{\omega}{2}\,\mathbf{bw}} \cdot n^{O(1)}$, where ω is the constant in the exponent of fast matrix multiplication (currently, $\omega \leq 2.376$). Exploiting planarity, we saw how to improve further upon

the existing bounds and gave a $3^{\mathbf{tw}} \cdot n^{O(1)}$ algorithm for Planar Dominating Set, representative for a number of improvements on results of the Tables 2.3 on Page 69 and 2.1 on Page 53 as shown in Table 2.2.

Table 2.2: Worst-case runtime expressed by treewidth **tw** and branchwidth **bw** of the input graph. In Table 2.1 on Page 53, only those graph problems are improved upon, which are unweighted or of small integer weights. Therefor, we state the improvements independently for weighted and unweighted graph problems. The second part of the table gives a summary of the most important improvements on exact and parameterized algorithms with parameter k. In some calculations, the fast matrix multiplication constant $\omega < 2.376$ is hidden.

	Previous results	New results
weighted Planar Dominating Set	$O(n2^{\min\{2\,\mathbf{tw},2.38\,\mathbf{bw}\}})$	$O(n2^{1.58\,\mathbf{tw}})$
unweighted Planar DS	—"—	$O(n2^{\min\{1.58\,\mathbf{tw},1.89\,\mathbf{bw}\}})$
w Planar Independent DS	$O(n2^{\min\{2\,\mathbf{tw},2.28\,\mathbf{bw}\}})$	$O(n2^{1.58\,\mathbf{tw}})$
uw Planar Independent DS	—"—	$O(n2^{\min\{1.58\,\mathbf{tw},1.89\,\mathbf{bw}\}})$
w Planar Total DS	$O(n2^{\min\{2.58\,\mathbf{tw},3\,\mathbf{bw}\}})$	$O(n2^{2\,\mathbf{tw}})$
uw Planar Total DS	—"—	$O(n2^{\min\{2\,\mathbf{tw},2.38\,\mathbf{bw}\}})$
w Planar Perfect Total DS	$O(n2^{\min\{2.58\,\mathbf{tw},3.16\,\mathbf{bw}\}})$	$O(n2^{\min\{2.32\,\mathbf{tw},3.16\,\mathbf{bw}\}})$
uw Planar Perfect Total DS	—"—	$O(n2^{\min\{2.32\,\mathbf{tw},2.53\,\mathbf{bw}\}})$
Planar Dominating Set	$O(2^{5.04\sqrt{n}})$ [50]	$O(2^{3.99\sqrt{n}})$
Planar Vertex Cover	$O(2^{3.18\sqrt{n}})$ [50]	$O(2^{2.52\sqrt{n}})$
Parameterized Planar DS	$O(2^{15.13\sqrt{k}}k + n^3)$ [49]	$O(2^{11.98\sqrt{k}}k + n^3)$
Parameterized Planar VC	$O(2^{5.67\sqrt{k}}k + n^3)$ [50]	$O(2^{3.56\sqrt{k}}k + n^3)$

2.4 Appendix

Other domination-type problems. For problems over vertex subsets having other domination-type constraints we get slightly different runtimes. Those are the other (σ, ϱ)-problems that are mentioned in Section 1.3. The papers [1, 7] considers these constraints, and give dynamic programming algorithms on nice tree-decompositions that take into account monotonicity properties to arrive at fast runtime.

Perfect code. We now describe the dynamic programming algorithms on semi-nice tree-decompositions *tree-decomposition!semi-nice tree-decomposition* for these problems, whose results are summarized in Table 2.3. We start by Perfect code , which is not an optimization problem, since any Perfect Code in a graph has the same size. However, it is NP-complete to decide if an arbitrary graph has a Perfect Code [68]. We construct an

algorithm using semi-nice tree-decompositions, using the same terminology and variables as in the DOMINATING SET algorithm.

We use the 3 vertex states **black**, **white** and **gray**. For index $s : X \to \{1, 0, 2\}$ the vertex subset S of G_X is *legal* for s if: S is a perfect code of $G_X \setminus X$ (i.e. every vertex $v \in V(G_X) \setminus X$ has $|N[v] \cap S| = 1$), vertices with state black are exactly $X \cap S$ and they have no neighbors with state black, vertices with state white have exactly one neighbor in $S \setminus X$, and vertices with state gray have zero neighbors in $S \setminus X$. Note that gray and white reflect only the number of dominating neighbors *outside* of the bag X. We sketch the further differences with the Dominating set algorithm.

A table index s stores True if there exists any vertex subset legal for s, and False otherwise. During Forget update we consider only indices in the child table $Table_C$ where the forgotten vertices are either in state black, or in state white and none of its neighbors in C are in state black, or in state gray and exactly one of its neighbors in C are in state black. During Root-optimization at X we look for an index storing True having the property that any vertex with state white has no neighbors in X with state black, and any vertex with state gray has exactly one neighbor in X with state black.

During the Join update the following 4 triples are necessary for vertices E and F: (black,black,black), (gray,gray,gray), (white,gray,white), (white,white,gray). The triple (white,white,white) was not necessary for the dominating set problem because of a monotonicity property between the white and gray states. We don't need the triple (white,white,white) since white reflects a dominating neighbor outside the bag X and Perfect Code asks for only one dominating neighbor. For the vertices in D we have 3 necessary triples, one for each vertex state. $Table_X(s)$ at a Join node X with children B, C is the disjunction of conjunctions $Table_B(s_b) \wedge Table_C(s_c)$ over all pairs of indices (s_b, s_c) such that the triple (s, s_b, s_c) is a necessary triple of indices. Then, set $Table_X(s) = False$ if for some new edge uv $s(u) = s(v) = 1$. The timing for the Join for the Perfect Code problem is thus $O(3^{|D|}4^{|E|+|F|})$, by a proof similar to the one for Lemma 2.1.5.

Total Dominating Set. We now turn to TOTAL DOMINATING SET where the vertices in the dominating set S also must have at least one neighbor in S. We use the 4 vertex states **black**, **white**, **blue**, and **gray**, where the two latter are temporary states. For index $s : X \to \{1, 0, \hat{1}, 2\}$ the vertex subset S of G_X is *legal* for s if: S is a total dominating set of $G_X \setminus X$, vertices with state black or blue are exactly $X \cap S$, and vertices with state black or white have a neighbor in S. Note that vertices with state blue and gray are simply constrained to be in S or not in S, respectively. Since these are also constraints on vertices with state black and white, respectively, we have the monotonicity property $Table_X(t) \leq Table_X(s)$ for pairs of indices t and s where $\forall v \in X$ either $t(v) = s(v)$ or $t(v) = \hat{1}$ and $s(v) = 1$ or $t(v) = 2$ and $s(v) = 0$.

The algorithm is very similar to the DOMINATING SET algorithm, except that also vertices in the dominating set have temporary state blue in addition to state black. Dur-

ing the Join update there are 6 triples necessary for Expensive vertices: (gray,gray,gray), (white,white,gray),(white,gray,white), (blue,blue,blue), (black,blue,black), and last but not least (black,black,blue). The first three triples occur also in the dominating set algorithm, while the argument that the three last triples for dominating vertices suffice is that the monotonicity property holds also between the black and blue states. Thus 6 triples total and runtime $O(6^k)$ for a Join update if all vertices are Expensive. Note that [7] incorrectly claims runtime $O(5^k)$, but this has been corrected to $O(6^k)$ in [1]. For the vertices in D we get 4 triples simply because we have 4 vertex states, while for vertices in F we get the 4 triples (white,white,gray),(white,gray,white) (black,black,blue), (black,blue,black) since gray and blue are temporary states. The timing for the Join for the Min Total Dominating set problem is thus $O(4^{|D|+|F|}6^{|E|})$.

Other variations. For Independent Dominating Set we get the same runtime as Dominating Set, while for Perfect Dominating Set we get the same runtime as Perfect Code. For Total Perfect Dominating Set we combine our solutions for Perfect Dominating Set and Total Dominating Set for a runtime for Join of $O(4^{|D|}5^{|F|}6^{|E|})$.

See column Join in Table 2.3 for an overview of these results. The previous best results for these problems [1, 7] correspond to our results when treating all vertices as Expensive, so we have moved closer to the goal of $\lambda^{|D|+|E|+|F|}$ time for a problem with λ vertex states. These algorithms can of course be extended also to more general (σ, ϱ)-sets. For example, if $\sigma = \{0, 1, ..., p\}$ and $\varrho = \{0, 1, ..., q\}$ we are asking for a subset $S \subseteq V(G)$ such that S induces a subgraph of maximum degree at most p with each vertex in $V(G) \setminus S$ having at most q neighbors in S. For this case we would use $p+q+2$ vertex states and get runtime $O((p + q + 2)^{|D|}(s(p) + s(q))^{|E|+|F|})$, where $s(i)$ is the number of pairs of ordered non-negative integers summing to i. Thus, for the Maximum 2-Packing problem (also known as Max Strong Stable set), which is of this form with $p = 0$ and $q = 1$, we get an $O(3^{|D|}4^{|E|+|F|})$ runtime for the Join operation.

The (k, r)-center problem. In the paper [32] the (k, r)-center problem is solved by dynamic programming on a branch-decomposition. The problems asks whether an input graph G has at most k vertices, called centers, such that every vertex is within distance at most r from some center. We now describe an algorithm solving the (k, r)-center problem on a semi-nice tree-decomposition by using as basis the algorithm in [32] on branch-decompositions. In the following we mainly describe the algorithm already given by [32], with the only addition to that algorithm being the handling of new edges in the Join update below. The problem asks whether an input graph G has $\leq k$ vertices, called *centers*, such that every vertex of G is within distance $\leq r$ from some center. We first transform a given branch-decomposition into a semi-nice tree-decomposition as described in Lemma 2.1.3. We can design a dynamic programming algorithm on semi-nice tree-decompositions that needs $2r + 1$ states with one state

defining centers, r states defining a vertex "having a center at distance i" for each $1 \leq i \leq r$, and r temporary states defining a vertex that "must get a center at distance i" for each $1 \leq i \leq r$ (this center will be reachable by a path through a not-yet considered neighbor.) These latter two types of states obey a monotonicity property similar to the white and gray states. The resulting algorithm will have $O((2r+1)^{|D|+|F|}(3r+1)^{|E|})$ time for updating Join nodes and $O(|X|(r+1)^{|F|}(2r+1)^{|X\setminus F|})$ for Introduce nodes X and $O((r+1)^{|C\setminus X|}(2r+1)^{|X|})$ for Forget node X and child C. On a Join node X with partition D, E, F consider the node t of the branch-decomposition guaranteed by Lemma 2.1.3 that has $E \subseteq mid(e) \cap mid(f) \cap mid(g)$ and $F \subseteq mid(f) \cap mid(g) \setminus mid(e)$ and $D \subseteq mid(e) \setminus mid(f) \cap mid(g)$. The first step of the Join update on node X with partition D, E, F can be derived from the update described in [32] for sets $X_3 = mid(e) \cap mid(f) \cap mid(g)$, $X_1 = mid(e) \cap mid(f) \setminus mid(g)$, $X_2 = mid(e) \cap mid(g) \setminus mid(f)$ and $X_4 = mid(f) \cap mid(g) \setminus mid(e)$. We then have to handle the new edges among vertices in D, but we can do that without increase of the runtime. After step 1 we have correct table entries for the graph $G_X \setminus New$, as in the proof of Lemma 2.1.5. To account for new edges we compute in step 2 and 3 New and $New(R)$ for each $R \subseteq D$ as in the Min Dom set algorithm, and then in step 4 we update in a loop from $i = 1..r$ each index s by $Table_X(s) = Table_X(s')$ where s' is defined by $s'(u) =$"must get a center at distance i" for any $u \in New(R)$ with $s(u) =$"having a center at distance i" and R being the vertices in D whose state in s is either "must get a center at distance $i-1$" or "having a center at distance $i-1$". Together with a straightforward extension of the update process on Introduce nodes and Forget nodes we obtain an algorithm on semi-nice tree-decompositions matching the runtime $O(2r+1)^{1.5\,\mathbf{bw}}$ of [32] when given a branch-decomposition of width **bw**.

See Table 2.3 for a summary of the results for each domination-type problem, expressed by **tw** and **bw** only, to not clutter the table, even though for each problem in the table such a cutoff point could be computed for which **pw** is best.

For the (k, r)-CENTER problem we get an algorithm with runtime $O(2r+1)^{1.5\,\mathbf{bw}}$, matching the time achieved by [32] for an algorithm working directly on the branch-decomposition by a similar argument as in Theorem 2.1.6.

Table 2.3: The number of vertex states λ and time for a Join operation with Expensive vertices E, Forgettable vertices F and Symmetric Difference vertices D. Worst-case runtime expressed also by treewidth **tw** and branchwidth **bw** of the input graph, and the cutoff point at which treewidth is the better choice. To not clutter the table, we leave out pathwidth **pw**, although for each problem there is a cutoff at which pathwidth would have been best.

	λ	Join	Total time	**tw** faster
Dominating Set	3	$O(3^{\lvert D\rvert+\lvert F\rvert}4^{\lvert E\rvert})$	$O(n2^{\min\{2\,\mathbf{tw},2.38\,\mathbf{bw}\}})$	$\mathbf{tw} \leq 1.19\,\mathbf{bw}$
Independent DS	3	$O(3^{\lvert D\rvert+\lvert F\rvert}4^{\lvert E\rvert})$	$O(n2^{\min\{2\,\mathbf{tw},2.38\,\mathbf{bw}\}})$	$\mathbf{tw} \leq 1.19\,\mathbf{bw}$
Perfect Code	3	$O(3^{\lvert D\rvert}4^{\lvert E\rvert+\lvert F\rvert})$	$O(n2^{\min\{2\,\mathbf{tw},2.58\,\mathbf{bw}\}})$	$\mathbf{tw} \leq 1.29\,\mathbf{bw}$
Perfect DS	3	$O(3^{\lvert D\rvert}4^{\lvert E\rvert+\lvert F\rvert})$	$O(n2^{\min\{2\,\mathbf{tw},2.58\,\mathbf{bw}\}})$	$\mathbf{tw} \leq 1.29\,\mathbf{bw}$
Max 2-Packing	3	$O(3^{\lvert D\rvert}4^{\lvert E\rvert+\lvert F\rvert})$	$O(n2^{\min\{2\,\mathbf{tw},2.58\,\mathbf{bw}\}})$	$\mathbf{tw} \leq 1.29\,\mathbf{bw}$
Total DS	4	$O(4^{\lvert D\rvert+\lvert F\rvert}6^{\lvert E\rvert})$	$O(n2^{\min\{2.58\,\mathbf{tw},3\,\mathbf{bw}\}})$	$\mathbf{tw} \leq 1.16\,\mathbf{bw}$
Perf Total DS	4	$O(4^{\lvert D\rvert}5^{\lvert F\rvert}6^{\lvert E\rvert})$	$O(n2^{\min\{2.58\,\mathbf{tw},3.16\,\mathbf{bw}\}})$	$\mathbf{tw} \leq 1.22\,\mathbf{bw}$

3 Employing structures for subexponential algorithms

All results of the previous chapter provide subexponential parameterized algorithms when property $\boxed{\Omega}$ of Section 1.4 holds. However, there are many parameterized problems for which there is no known algorithm providing property $\boxed{\Omega}$ in general. The typical running times of dynamic programming algorithms for these problems are $O(\mathbf{bw}(G)!) \cdot n^{O(1)}$, $O(\mathbf{bw}(G)^{\mathbf{bw}(G)}) \cdot n^{O(1)}$, or even $O(2^{\mathbf{bw}(G)^2}) \cdot n^{O(1)}$. Examples of such problems are k-Longest Path, k-Feedback Vertex Set, k-Connected Dominating Set, and k-Graph TSP. Usually, these are problems in NP whose certificate verifications involves some connectivity question. In this section, we show that for such problems one can prove that $\boxed{\Omega}$ actually holds for the graph class that we are interested in. To do this, one has to make further use of the structural properties of the class (again from the Graph Minors Theory) that can vary from planar graphs to H-minor-free graphs. In other words, we use the structure of the graph class not only for proving $\boxed{\Lambda}$ (see Section 1.4) but also for proving $\boxed{\Omega}$.

3.1 Connectivity and surfaces

Vertex- and edge subset problems can be solved on graphs of bounded treewidth by using dynamic programming techniques, as we saw in Chapter 2, though for edge subset problems, there is an extra cost in computation due to the connectivity. For graphs embedded on surfaces (as well as for graphs excluding some fixed sized minor), We will see in the later sections of this chapter, that one can employ the *Catalan structure* of the graph embedding and its decomposition to check solutions and partial solutions in a more efficient way. The basic idea is that the solution consists of drawings of disjoint components or paths of a plane graph. That is, if one visualizes components by polygons and paths by arcs in the plane, they do not cross. In addition to this, we have that all polygons or arcs are connected to a cycle which visualizes a noose of small length ℓ. Then the number of solutions stands in direct relation to the *Catalan number* of ℓ, what gives us a new upper bound.

3.1.1 Components and paths

Let G be a graph and let $E \subseteq E(G)$ and $S \subseteq V(G)$. We consider edge-subset problems, whose solution consists of one or several connected components. In order to count the number of states at each step of dynamic programming, we have to estimate the number of collections of internally vertex-disjoint components on the induced subgraph $G[E]$ of

G. We count those components which use edges from E, intersect with vertices of S and own a problem specific property $\mathfrak{P}$. In particular, $\mathfrak{P}$ defines the properties of the components in $G[E]$ that are necessary for a valid solution to the problem. We denote the term "component C has property $\mathfrak{P}$" by $C(\mathfrak{P})$. $\mathfrak{P}$ can give several properties and is defined by property **G** of Section 1.3.2 of the respective edge-subset problem. It could mean for example, that each component is acyclic, or that every vertex of a component C in $G[E]$ has an even number of adjacent vertices in C.

Definition 3.1.1. *Let* $\mathbf{C}$ *be a collection of vertex-disjoint components* C *in* G, $E \subseteq E(G)$ *and* $S \subseteq V(G)$.

$$\mathbf{comp}_G(E, S, \mathfrak{P}) := \{\mathbf{C} \mid \forall C \in \mathbf{C} : E(C) \cap E(G) \subseteq E \wedge C \cap S \neq \varnothing \wedge C(\mathfrak{P})\}$$

is the set of all collections of components on set E *intersecting with* S *and having property* $\mathfrak{P}$.

Let $\mu : \mathbf{C} \to \mathbf{C}$ *be a bijection. Define equivalence relation* $'\sim'$ *on* $\mathbf{comp}_G(E, S, \mathfrak{P})$*:*

For $\mathbf{C}_1, \mathbf{C}_2 \in \mathbf{comp}_G(E, S, \mathfrak{P})$: $\mathbf{C}_1 \sim \mathbf{C}_2$, *if* $\forall C_1 \in \mathbf{C}_1, C_2 \in \mathbf{C}_2$ *with* $\mu(C_1) = C_2$ *we have that* $C_1 \cap S = C_2 \cap S$.

Denote by

$$\texttt{q-comp}_G(E, S, \mathfrak{P}) := |\mathbf{comp}_G(E, S, \mathfrak{P})/\sim|$$

the cardinality of the quotient set of $\mathbf{comp}_G(E, S, \mathfrak{P})$ *by* $\sim$.

If $\mathfrak{P}$:= "arbitrary", we simply use the shorter notion $\mathbf{comp}_G(E, S)$. Note that $\mathbf{comp}_G(E, S) \subseteq \mathbf{comp}_G(E, S, \mathfrak{P})$ for any property $\mathfrak{P}$.

For edge-subset problems whose solution is a cycle or path, we have that $\mathfrak{P}_P$:= "every component is a path with both endpoints in S".

Definition 3.1.2. *For a collection of vertex-disjoint paths in* $G[E]$ *with endpoints in* S, *where* $E \subseteq E(G)$ *and* $S \subseteq V(G)$, *define*

$$\mathbf{paths}_G(E, S) := \mathbf{comp}_G(E, S, \mathfrak{P}_P)$$

and

$$\texttt{q-paths}_G(E, S) = |\mathbf{paths}_G(E, S)/\sim|.$$

3.1.2 Catalan structures

Given a graph G and let $S \subseteq V(G)$ be a separator of G. Let G_1 and G_2 be subgraphs of G such that $G_1 \cup G_2 = G$, $V(G_1) \cap V(G_2) = S$ and $E(G_1) \cap E(G_2) = \varnothing$.

We say that S has *Catalan structure* if

$$\texttt{q-comp}_{G_i}(E(G_i), S) = 2^{O(|S|)}, \text{ for } i = 1, 2. \tag{3.1}$$

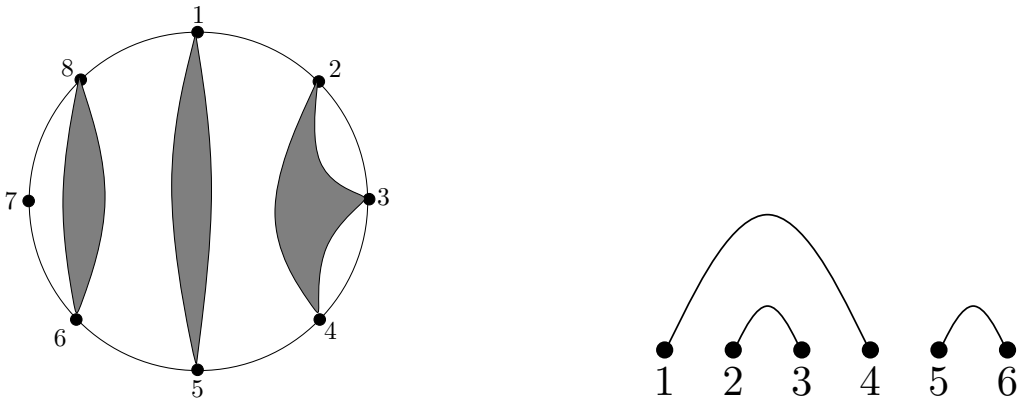


Figure 3.1: The left diagram shows a non-crossing partition, the right a non-crossing matching.

Branch-decomposition with Catalan structure. The reason we define $\texttt{q-comp}_G(E,S)$ is because the number of states for $e \in E(T)$ is bounded by $O(\texttt{q-comp}_{G_i}(E(G_i), \text{mid}(e)))$, when applying dynamic programming on some middle set $\text{mid}(e)$ of the branch decomposition (T,μ).

Given a graph G and a branch decomposition (T,μ) of G, we say that (T,μ) has *Catalan structure* if for every edge $e \in E(T)$ and any $i \in \{1,2\}$,

$$\texttt{q-comp}_{G_i}(E(G_i), \text{mid}(e)) = 2^{O(\mathbf{bw}(T,\mu))}. \tag{3.2}$$

3.1.3 Non-crossing matchings

To estimate $\texttt{q-comp}_G(E,S)$ or $\texttt{q-paths}_G(E,S)$ in a graph embedded onto a surface when S is cyclic ordered, that is, when the vertices of S can be connected by a noose, we need the following lemmas.

Non-crossing partitions. Non-crossing partitions give us the key to our later dynamic programming approach. A *non-crossing partition (ncp)* is a partition $P(n) = \{P_1, \ldots, P_m\}$ of the set $S = \{1, \ldots, n\}$ such that there are no numbers $a < b < c < d$ where $a, c \in P_i$, and $b, d \in P_j$ with $i \neq j$. A partition can be visualized by a circle with n equidistant vertices on its border, where every set of the partition is represented by the convex polygon with its elements as endpoints. A partition is non-crossing if these polygons do not overlap. See the left diagram in Figure 3.1 for an example.

Non-crossing partitions were introduced by Kreweras [70], who showed that the number of non-crossing partitions over n vertices is equal to the n-th Catalan number:

$$\text{CN}(n) = \frac{1}{n+1}\binom{2n}{n} \sim \frac{4^n}{\sqrt{\pi} n^{\frac{3}{2}}} \approx 4^n \tag{3.3}$$

Thus, for a graph G embedded onto a sphere $\mathbb{S}_0$ and a vertex set S with cardinality ℓ, that is connected by a noose, we have that

$$\texttt{q-comp}_G(E,S) = \mathrm{CN}(\ell) = O(4^\ell).$$

Non-crossing matchings.

Lemma 3.1.3. *[70] Let $P(n) = \{P_1, \ldots, P_{\frac{n}{2}}\}$ be a partition of an ordered set $S = \{x_1, \ldots, x_n\}$ into tuples, such that there are no elements $x_i < x_j < x_k < x_\ell$ with $\{x_i, x_k\}$ and $\{x_j, x_\ell\}$ in $P(n)$. Let $\mathcal{P}_S$ be the collection of all such partitions of S. Then,*

$$|\mathcal{P}_S| = O(2^n).$$

The partitions of Lemma 3.1.3 are called *non-crossing matchings*. A non-crossing matching can be visualized by placing n vertices on a cycle , and connecting matching vertices by arcs at one side of the cycle. See the right diagram in Figure 3.1 for an example. In a graph G, each element P of $\mathbf{paths}_G(E,S)/\sim$ can be seen as set of arcs with endpoints in S. If every P is a non-crossing matching, we say that the paths in $P_i \in \mathbf{paths}_G(E,S)$ with $P \sim P_i$ are *non-crossing* and S has a Catalan structure.

Thus, for a graph G embedded onto a sphere $\mathbb{S}_0$ and a vertex set S with cardinality ℓ, that is connected by a noose, we have that

$$\texttt{q-paths}_G(E,S) = |\mathcal{P}_S| = O(2^\ell).$$

(n,r)-non-crossing matchings. The following lemma gives an estimation on the non-crossing matchings arising from the case that one has several cycles in the plane that are connected by arcs. This will be needed for dynamic programming in Chapter 3 for the case of graphs of bounded genus. After planarizing the graph by cutting along noncontractible nooses , the situation arises, that components and paths, respectively have their endpoints in several such "cut-nooses".

Lemma 3.1.4. *Let r disjoint empty discs $\Delta_1, \ldots, \Delta_r$ be embedded on the sphere $\mathbb{S}_0$ where each disc is bounded by a cycle of at most n vertices. Let P be a set of arcs connecting the vertices, such that P can be embedded onto $\mathbb{S}_0 - \{\Delta_1, \ldots, \Delta_r\}$ without arcs crossing. Let $\mathcal{P}_{n,r}$ be the collection of all such P. Then,*

$$|\mathcal{P}_{n,r}| \leq r^{r-2} \cdot n^{2r} \cdot 2^{rn}.$$

Proof. We show how to reduce the counting of $|\mathcal{P}_{n,r}|$ to non-crossing matchings. Here we deal with several open disks and our intention is to transform them into one single disk in order to apply Lemma 3.1.3.

First lets assume $r = 2$. Choose a set $P \in \mathcal{P}_{n,2}$. Assume two vertices x and y on the boundary of two different disks Δ_1 and Δ_2 in being two endpoints of an arc $\{x,y\}$ in P. We observe that no other arc in P crosses $\{x,y\}$ in the $\mathbb{S}_0$-embedding of P. So

we are able to 'cut' the sphere $\mathbb{S}_0$ along $\{x, y\}$ and, that way, create a "tunnel" between Δ_1 and Δ_2 unifying them to a single disk and thus reduced the problem to counting non-crossing matchings. That is, for obtaining a rough upper bound on $|\mathcal{P}_{n,2}|$, one fixes every pair of vertices $x \in \Delta_1$ and $y \in \Delta_2$ and we obtain

$$|\mathcal{P}_{n,2}| = O(n^2 \cdot 2^{2n}).$$

The next difficulty is that all disks in $\Delta_1, \ldots, \Delta_r$ are connected by arcs of $P \in \mathcal{P}_{n,r}$ in an arbitrary way. We use a tree structure in order to cut the sphere along that structure. Given such a tree structure, we create tunnels in order to connect the open disks and to merge them to one disk.

Consider all $\leq n^{n-2}$ possible spanning trees on n vertices [20]. Here, we have a spanning tree over r vertices, representing the r disks in $\Delta_1, \ldots, \Delta_r$. Then the boundary of each disk has length $\leq n$. Hence, there are $O(n^2)$ possible fixed arcs between the boundaries of each two disks. Then we obtain a rough upper bound of n^{2r} on the number of possible fixed arcs between the disks in a given tree-structure. We obtain $r^{r-2} \cdot n^{2r}$ possibilities for above concatenation and tunneling of $\Delta_1, \ldots, \Delta_r$. We argue that P has a Catalan structure when tunneling the disks in this way. Thus,

$$|\mathcal{P}_{n,r}| \leq r^{r-2} \cdot n^{2r} \cdot 2^{rn}.$$

□

We call an element of $\mathcal{P}_{n,r}$ a (n, r)-*non-crossing matching*. If for a graph G, each element of $\mathbf{paths}_G(E, S)/\sim$ is a (n, r)-non-crossing matching then S has Catalan structure.

h-almost-(n, r)-non-crossing matchings. This lemma will also be needed for dynamic programming in Chapter 3 for solving connected problems on H-minor-free graphs, where we obtain an extended case of the bounded genus graph with additional h "areas of non-planarity", so-called *vortices*.

Lemma 3.1.5. *Let r disjoint empty discs $\Delta_1, \ldots, \Delta_r$ be embedded on the sphere $\mathbb{S}_0$ where each disc is bounded by a cycle of at most n vertices. Let $\mathbb{S}_0 - \{\overline{\Delta}_1, \ldots, \overline{\Delta}_r\}$ contain $\leq h$ disjoint discs $R_1, \ldots, R_h$.*

Let P be a set of arcs connecting the vertices of $\mathbf{bor}(\Delta_1), \ldots, \mathbf{bor}(\Delta_r)$, such that P can be embedded onto $\mathbb{S}_0 - \{\Delta_1, \ldots, \Delta_r\}$ with arcs crossing only inside $R_1, \ldots, R_h$ such that $P \cap R_j$ $(1 \leq j \leq h)$ is a superposition of h non-crossing matchings. Then P is a superposition of $O((h + r)^h)$ many (n, r)-non-crossing matchings. Let $\mathcal{P}^h_{n,r}$ be the collection of all such P. Thus,

$$|\mathcal{P}^h_{n,r}| \leq (r^{r-2} \cdot n^{2r} \cdot 2^{rn})^{(h+r)^h}.$$

Proof. We observe that only h arcs of $P \cap R_j$ may cross mutually inside one of R_j ($1 \leq j \leq h$), but since the entire arc α of P may enter and leave R_j arbitrarily often, we may have more than h mutually crossings of P in R_j. However, we observe that α then cuts the $\mathbb{S}_0 - \{\Delta_1, \ldots, \Delta_r\}$ into several discs. It follows, together with the Helly property of circular arcs, that there are roughly $\frac{3}{2}h + r - 1$ arcs that mutually cross in R_j. We color the arcs of P such that no two arcs of the same color class cross. For arcs crossing in one R_j we thus need up to $\frac{3}{2}h + r - 1$ colors.

Furthermore, we observe that two arcs of P may be assigned with the same color in R_j but cross in another R_i, etc. Hence, we have a rough upper bound of $(h + r)^h$ colors, that is every arc can be assigned by $\frac{3}{2}h + r - 1$ colors per R_j and is thus assigned by a h-vector of colors for all R_j ($1 \leq j \leq h$). With Lemma 3.1.4, we count for every color class the number of (n, r)-non-crossing matchings and we get that the overall size of $\mathcal{P}^h_{n,r}$ is bounded by $(r^{r-2} \cdot n^{2r} \cdot 2^{rn})^{(h+r)^h}$.

□

Both, Lemma 3.1.4 and Lemma 3.1.5 can be extended to non-crossing partitions and thus give an upper bound on $\texttt{q-comp}_G(E, S)$ on graphs of bounded genus and H-minor-free graphs (that is h-almost embeddable graphs), respectively.

Planar graphs. In Section 3.2, we introduce a $2^{O(\mathbf{bw}(T,\mu,\pi))}n^{O(1)}$ algorithm for the k-Planar Longest Path. One may use k-Longest path as an exemplar for other problems of the same nature.

We will see in Section 3.2 that the application of dynamic programming on an sc-decomposition (T, μ, π) is the $2^{O(\mathbf{bw}(T,\mu,\pi))}n^{O(1)}$ algorithm for proving property $\boxed{\Omega}$ for planar graphs. By further improving the way the members of $\texttt{q-paths}_{G_i}(E(G_i), \text{mid}(e))$ are encoded during this procedure, one can bound the hidden constants in the big-O notation on the exponent of this algorithm. For example, for Planar k-Longest Path $\beta \leq 2.63$. With analogous structures and arguments it follows that for Planar k-Graph TSP $\beta \leq 3.84$, for Planar k-Connected Dominating Set $\beta \leq 3.82$, for Planar k-Feedback Vertex Set $\beta \leq 3.56$.

We also will give algorithms for the exact variants of these problems, and some non-parameterized problems, such as Planar Hamiltonian Path, Minimum Cycle Cover and Steiner Tree.

Bounded genus. In Section 3.3 and 3.4, the results on exact algorithms are generalized for graphs with genus one and bounded genus, respectively (now constants for each problem depend also on the genus). This generalization requires a suitable "bounded genus"-extension of the Catalan structure notion. We introduce two different approaches to attack these problems. The idea behind the first approach is to planarize the graph

and reformulate the original problem on the planarized graph, that is, we get subexponentially many subproblems each of which we solve with dynamic programming in subexponential time.

The second approach is applied in Section 3.5 as part of a parameterized algorithm for finding k-Longest Path on H-minor-free graphs.

Excluding a minor. The final step is to prove property $\boxed{\Omega}$ for H-minor-free graphs. Let $\mathcal{G}$ be a graph class excluding some fixed graph H as a minor. In Section 3.5, we will see that every graph $G \in \mathcal{G}$ with $\mathbf{bw}(G) \leq \ell$ has an branch decomposition of width $O(\ell)$ with the Catalan structure and that such a decomposition can be constructed in $f(|H|) \cdot n^{O(1)}$ steps, where f is a function depending only on H.

We will give an algorithm constructing the claimed branch decomposition using a structural characterization of H-minor-free graphs, given in [85]. Briefly, any H-minor-free graph can be seen as the result of gluing together (identifying constant size cliques and, possibly, removing some of their edges) graphs that, after the removal of some constant number of vertices (called *apices*) can be "almost" embedded in a surface of constant genus. Here, by "almost" we mean that we permit a constant number of non-embedded parts (called *vortices*) that are "attached" around empty disks of the embedded part and have a path-like structure of constant width. The algorithm has several phases, each dealing with some level of this characterization, where an analogue of sc-decomposition for planar graphs is used. The core of the correctness proof is based on the fact that the structure of the embeddable parts of this characterization (along with vortices) is "close enough" to be plane, so to roughly maintain the Catalan structure property.

The algorithm implies $\boxed{\Omega}$ for k-Longest Path on H-minor-free graphs.

3.2 Planar graphs

In what follows, we give a $2^{O(\mathbf{bw}(T,\mu,\pi))}n^{O(1)}$ algorithm for the k-Planar Longest Path. We use the k-Planar Longest path algorithm as a black box for algorithms for other problems of the same nature.

3.2.1 k-Planar Longest Path

In Section 1.4, property $\boxed{\Omega}$ holds for edge subset problems (with global properties) on planar graphs because of the following combinatorial result using Catalan structures and non-crossing matchings of Section 3.1. In particular, it holds that every sc-decomposition has the Catalan structures.

Theorem 3.2.1. *Every planar graph has an optimal branch-decomposition with the Catalan structure that can be constructed in polynomial time.*

Proof. Compute a sc-decomposition (T, μ, π) as in Subsection 2.3.1 (constructed using the polynomial algorithm in [89]). Let O_e be a noose meeting some middle set $\mathrm{mid}(e)$ of (T, μ, π). Let us count in how many ways this noose can cut paths of G. Observe that each path is cut into at most $\mathbf{bw}(T, \mu, \pi)$ parts. Each such part is itself a path whose endpoints are pairs of vertices in O_e. Notice also that, because of planarity, no two such pairs can cross. By Lemma 3.1.3, counting the ways O_e can intersect paths of G is equivalent to counting non-crossing matchings in a cycle (the noose) of length $\mathbf{bw}(T, \mu, \pi)$ which, in turn, is bounded by the Catalan number of $\mathbf{bw}(T, \mu, \pi)$ that is $2^{O(\mathbf{bw}(T,\mu,\pi))}$.

□

The k-Planar Longest Path problem asks for a given planar graph G to find a path of length at least a parameter k. One can also ask for the k-Planar Longest Cycle, we will see how to solve in the same way.

Before we proceed, let us recall the notion of a minor in Section 1.2. Let us note that if a graph H is a minor of G and G contains a path of length at least k, then so does G.

Recall that according to Theorem 1.4.2, for a positive integer ℓ, every planar graph with no $(\ell \times \ell)$-grid as a minor has branchwidth at most 4ℓ.

The main result of the section is the following.

Theorem 3.2.2. *For every planar graph G and any positive integer ℓ, it is possible to construct a time $n^{O(1)}$ algorithm that outputs one of the following:*

1. *A correct report that G contains a $(\ell \times \ell)$-grid as a minor.*
2. *A branch-decomposition (T, τ) with the Catalan structure and of width $O(\ell)$.*

The algorithm works as follows: For a planar input graph G and parameter k, first compute the branchwidth of G and the corresponding sc-decomposition (T, μ, π) by using the algorithm of [89]. If the branchwidth of G is at least $4\sqrt{k+1}$ then by Theorem 1.4.2, G contains a $(\sqrt{k+1} \times \sqrt{k+1})$-grid as a minor. Thus G contains a path of length at least k, since every $(\sqrt{k} \times \sqrt{k})$-grid, $k \geq 2$, contains a path of length at least $k-1$. If the branchwidth of G is less than $4\sqrt{k+1}$, we use standard dynamic programming on the sc-decomposition (T, μ, π) as described in Section 2.1. Since (T, μ, π) has the Catalan structure, we can solve the problem in time $2^{O(\sqrt{k})}\, n + n^{O(1)}$.

However, we are able to give a faster algorithm by using an efficient encoding and techniques introduced in Section 2.2. In what follows, we will see a detailed description of such encoding that we will reuse for several other edge-subset problems and prove the following:

Theorem 3.2.3. k-Planar Longest Path *is solvable in time $O(2^{10.984\sqrt{k}} n^{3/2} + n^3)$.*

Encoding of Catalan structures

Here we study how sc-decompositions in combination with non-crossing matching can be used to design subexponential algorithms.

Let us reformulate the k-Planar Longest Path problem in a different way: A labeling $\mathcal{H} : E(G) \to \{0,1\}$ is *k-long* if the subgraph $G_{\mathcal{H}}$ of G formed by the edges with label '1' is a path of length at least k. Find a k-long labeling $\mathcal{H}$ that is a labeling with $\sum_{e \in E(G)} \mathcal{H}(e) \geq k$. For an edge labeling $\mathcal{H}$ and a vertex $v \in V(G)$ we define the *$\mathcal{H}$-degree* $deg_{\mathcal{H}}(v)$ of v as the sum of labels assigned to the edges incident to v. Let (T, μ, π) be a sc-decomposition of G of width ℓ. We root T as in Section 2.1 at a node r.

For every edge e of T the subtree toward the leaves is called the *lower part* and the rest the *residual part* with regard to e. We call the subgraph G_e induced by the leaves of the lower part of e the *subgraph rooted at e*. Let e be an edge of T and let O_e be the corresponding noose in Σ. The noose O_e partitions Σ into two discs, one of which, Δ_e, contains G_e.

Given a labeling $\mathcal{P}[e]\colon E(G_e) \to \{0,1\}$ for every edge in G_e, we define for every vertex v in G_e the *$\mathcal{P}[e]$-degree* $deg_{\mathcal{P}[e]}(v)$ to be the sum of the labels on the edges incident to v. Let $G_{\mathcal{P}[e]}$ be the subgraph induced by the edges with label '1'. We call $\mathcal{P}[e]$ a *partial k-long labeling* if $G_{\mathcal{P}[e]}$ satisfies the following properties:

- For every vertex $v \in V(G_e) \setminus O_e$, $deg_{\mathcal{P}[e]}(v) \in \{0,2\}$.
- Every connected component of $G_{\mathcal{P}[e]}$ has exactly two vertices in the noose O_e with $deg_{\mathcal{P}[e]}(v) = 1$, all other vertices of $G_{\mathcal{P}[e]}$ have $deg_{\mathcal{P}[e]}(v) = 2$.

Observe that $G_{\mathcal{P}[e]}$ forms a collection of disjoint paths with endpoints in O_e, and note that every partial k-long labeling of $G_{\{r,s\}}$ forms a k-long labeling.

Due to Section 3.1, define the set of all collections $G_{\mathcal{P}[e]}$ as $\mathbf{paths}_{G_e}(E(G_e), V(O_e))$ and because of sc-decompositions having Catalan structure, we get

$$\texttt{q-paths}_{G_e}(E(G_e), V(O_e)) = 2^{O(|V(O_e)|)}.$$

For a partial k-long labeling $\mathcal{P}[e]$ let P be a path of $G_{\mathcal{P}[e]}$. As the graph is planar, no paths cross and we can reduce P to an arc in Δ_e with endpoints on the noose O_e. If we do so for all paths, the endpoints of these arcs form a non-crossing matching.

Because O_e bounds the disc Δ_e and the graph $G_{\mathcal{P}[e]}$ is in Δ_e, we can to scan the vertices of $V(P) \cap O_e$ according to the ordering π and mark with '$1_[$' the first and with '$1_]$' the last vertex of P on O_e. Mark the endpoints of all paths of $G_{\mathcal{P}[e]}$ in such a way. Then the obtained sequence with marks '$1_[$' and '$1_]$', decodes the complete information on how the endpoints of $V(G_{\mathcal{P}[e]})$ hit O_e. With the given ordering π the '$1_[$' and '$1_]$' encode a non-crossing matching. The other vertices of $V(G_{\mathcal{P}[e]}) \cap O_e$ are the 'inner' vertices of the paths and we mark them by '2'. All vertices of O_e that are not in $G_{\mathcal{P}[e]}$ are marked by '0'.

For an edge e of T and the corresponding noose O_e, the state of dynamic program-

ming is specified by an ordered ℓ-tuple $\vec{t_e} := (v_1, \ldots, v_\ell)$. Here, the variables $v_1, \ldots, v_\ell$ correspond to the vertices of $O_e \cap V(G)$ taken according to the cyclic order π with an arbitrary first vertex. This order is necessary for a well-defined encoding where the variables v_i take one of the four values: 0, $1_[$, $1_]$, 2. Hence, there are at most $O(4^\ell |V(G)|)$ states. For every state, we compute a value $W_e(v_1, \ldots, v_\ell)$ that is the maximum length over all partial k-long labelings $\mathcal{P}[e]$ encoded by $v_1, \ldots, v_\ell$. If no such labeling exists we have $W_e(v_1, \ldots, v_\ell) = -\infty$.

For an illustration of a partial k-long labeling see Figure 3.2.

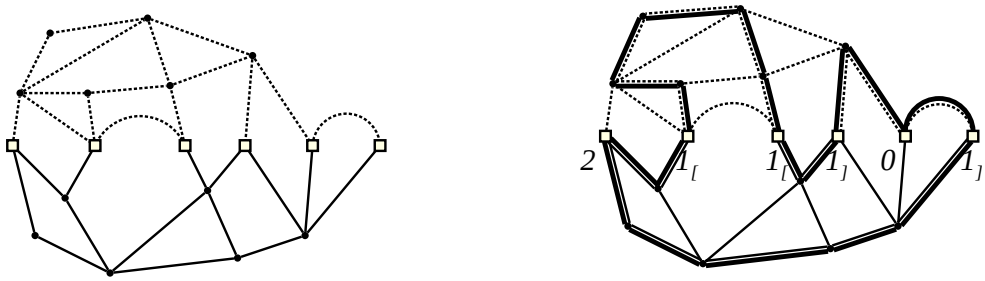


Figure 3.2: On the left we see a graph G partitioned by the rectangle vertices of $O_e \cap V(G)$ into G_e in drawn-through edges and $\overline{G_e}$ in dashed edges. On the right subgraph $G_\mathcal{H}$ marks a k-long path. $G_\mathcal{H}$ is partitioned by the vertices of $O_e \cap V(G)$ which are labeled corresponding to two vertex-disjoint paths in G_e induced by the partial k-long labeling $\mathcal{P}[e]$.

To compute an k-long labeling we perform dynamic programming over middle sets $\text{mid}(e) = O(e) \cap V(G)$, starting at the leaves of T and working bottom-up toward the root edge. The first step in processing the middle sets is to initialize the leaves with values $W_e(0,0) = 0$, and $W_e(1_[, 1_]) = 1$, depending on the edge of the graph corresponding to the leaf be in a partial k-long labeling or not. Then, bottom-up, update every pair of states of two child edges e_L and e_R to a state of the parent edge e_P assigning a finite value W_P if the state corresponds to a feasible partial k-long labeling.

Let O_L, O_R, and O_P be the nooses corresponding to edges e_L, e_R and e_P, and let Δ_L, Δ_R and Δ_P be the discs bounded by these nooses. Recall from Subsection 2.3.2 how Intersection-, Forget and Symmetric difference vertices relate with the nooses O_L, O_R, and O_P.

We compute all valid assignments to the variables $\vec{t_P} = (v_1, v_2, \ldots, v_p)$ corresponding to the vertices $\text{mid}(e_P)$ from all possible valid assignments to the variables of $\vec{t_L}$ and $\vec{t_R}$. For a symbol $x \in \{0, 1_[, 1_], 2\}$, we denote by $|x|$ its 'numerical' part, e.g. $|1_[| = 1$. We say that an assignment c_P is *formed* by assignments c_L and c_R if for every vertex

$v \in (O_L \cup O_R \cup O_P) \cap V(G)$:

- $\underline{v \in \{L, R\}}$: $c_P(v) = c_L(v)$ if $v \in O_L \cap V(G)$, and $c_P(v) = c_R(v)$ otherwise.
- $\underline{v \in F}$: $|c_L(v)| + |c_R(v)| \in \{0, 2\}$.
- $\underline{v \in I}$: $|c_P(v)| = |c_L(v)| + |c_R(v)| \leq 2$.

We compute all ℓ-tuples for $\mathrm{mid}(e_P)$ that can be formed by tuples corresponding to $\mathrm{mid}(e_L)$ and $\mathrm{mid}(e_R)$ and check if the obtained assignment corresponds to a labeling without cycles. For every encoding of $\vec{t}_P$, we set $W_P = \max\{W_P, W_L + W_R\}$.

For the root edge $\{r, s\}$ and its children e' and e'' note that $(O_{e'} \cup O_{e''}) \cap V(G) = F$ and $O_{\{r,s\}} = \varnothing$. Hence, for every $v \in V(G_{\mathcal{P}[\{r,s\}]})$ it must hold that $deg_{\mathcal{P}[\{r,s\}]}(v)$ is two (except for the two endpoints of the entire path), and that the labeling form a path of length $\geq k$. The k-long labeling of G results from $max_{\vec{t}_{\{r,s\}}}\{W_r\}$.

Analyzing the algorithm, we obtain the following lemma.

Lemma 3.2.4. *k-Planar Longest Path on a planar graph G with branchwidth ℓ can be solved in time $O(2^{3.404\ell}\ell n + n^3)$.*

Proof. By Theorem 2.3.1, an sc-decomposition $\langle T, \mu, \pi \rangle$ of width at most ℓ of G can be found in $O(n^3)$.

For a worst-case scenario, assume we have three adjacent edges e_P, e_L, and e_R of T with $|O_L| = |O_R| = |O_P| = \ell$. Without loss of generality we limit our analysis to even values for ℓ, and assume there are no intersection vertices. This can only occur if $|F| = |L \cap O_L| = |R \cap O_R| = \frac{\ell}{2}$.

By just checking every combination of ℓ-tuples from O_L and O_R we obtain a bound of $O(\ell 4^{2\ell})$ for our algorithm.

Some further improvement is apparent, as for the vertices $u \in F$ we want the sum of the $\{0, 1_[, 1_], 2\}$ assignments from both sides to be 0 or 2, i.e., $|c_L(u)| + |c_R(u)| \in \{0, 2\}$.

We start by giving an expression for $Q(\ell, m)$: the number of ℓ-tuples over ℓ vertices where the $\{1_[, 1_]\}$ assignments for m vertices from F is fixed. The only freedom is thus in the $\ell/2$ vertices in $L \cap O_L$ and $R \cap O_R$, respectively:

$$\mathrm{Q}(\ell, m) = \sum_{i=0}^{\frac{\ell}{2}} \binom{\frac{\ell}{2}}{i} 2^{\frac{\ell}{2}-i} \mathrm{M}(i + m) \tag{3.4}$$

This expression is a summation over the number of $1_[$'s and $1_]$'s in $L \cap O_L$ and $R \cap O_R$, respectively. The term $\binom{\frac{\ell}{2}}{i}$ counts the possible locations for the $1_[$'s and $1_]$'s, the $2^{\frac{\ell}{2}-i}$ counts the assignment of $\{0, 2\}$ to the remaining $\ell/2 - i$ vertices, and the $\mathrm{M}(i+m)$ term counts the non-crossing matchings over the $1_[$'s and $1_]$'s. As we are interested in exponential behavior for large values of ℓ we ignore if $i + m$ is odd, and use that

$M(n) \approx 2^n$:

$$Q(\ell, m) = O(\sum_{i=0}^{\frac{\ell}{2}} \binom{\frac{\ell}{2}}{i} 2^{\frac{\ell}{2}-i} 2^{i+m}) = O(2^{\ell+m}) \tag{3.5}$$

We define $C(\ell)$ as the number of possibilities of forming an ℓ-tuple from O_P. We sum over i: the number of $1_[$'s and $1_]$'s in the assignment for F:

$$C(\ell) = \sum_{i=0}^{\frac{\ell}{2}} \binom{\frac{\ell}{2}}{i} 3^{\frac{\ell}{2}-i} Q(\ell, i)^2 = O(\sum_{i=0}^{\frac{\ell}{2}} \binom{\frac{\ell}{2}}{i} 3^{\frac{\ell}{2}-i} 2^{2\ell} 2^{2i}) \tag{3.6}$$

The term $3^{\frac{\ell}{2}-i}$ counts the number of ways how the vertices of F are assigned with 0 and 2. The term 2^{2i} counts for F the number of ways vertices are assigned on both side by symbols with numerical value one. Straightforward calculation yields:

$$C(\ell) = O(3^{\frac{\ell}{2}} \cdot 4^{\ell} \sum_{i=0}^{\frac{\ell}{2}} \binom{\frac{\ell}{2}}{i} (\frac{4}{3})^i) = O(3^{\frac{\ell}{2}} 4^{\ell} (\frac{7}{3})^{\frac{\ell}{2}}) = O((4\sqrt{7})^{\ell}) \tag{3.7}$$

Since we can check in time linear in ℓ if an assignment forms no cycle and the number of edges in the tree of a branch-decomposition is $O(n)$, we obtain an overall running time of $O((4\sqrt{7})^{\ell}\ell n + n^3) = O(2^{3.404\ell}\ell n + n^3)$. □

Forbidding Cycles

We can further improve upon the previous bound by only forming encodings that do not create a partial cycle. As cycles can only be formed at the vertices in F with numerical part 1 in both O_L and O_R, we only consider these vertices. Note that the previous postprocessing step uncovered forbidden solutions. Thus, this observation helps, so that we replace the latter step by a preprocessing step and remove the states for the F-set that are forbidden.

Lemma 3.2.5. *k-Planar Longest Path on a planar graph G with branchwidth ℓ can be solved in time $O(2^{3.37\ell}\ell n + n^3)$.*

Proof. We would like to have an upper bound for the number of combinations from O_L and O_R that do not induce a cycle. This bound could then be applied to the previous analysis.

Let F have n vertices and be assigned by an ordered n-tuple of variables $(v_1, \ldots, v_n)$. Each variable v_i is a two-tuple $(c_L(v_i), c_R(v_i))$ of assignments $c_L, c_R \in \{1_[, 1_]\}$ of vertex v_i such that $|c_L(v_i)| + |c_R(v_i)| = 2$. For example, suppose F has only two vertices x and y. A cycle is formed if $c_L(x) = c_R(x) = 1_[$ and $c_L(y) = c_R(y) = 1_]$. That is, $((1_[, 1_[), (1_], 1_]))$ encodes a cycle. Let $B(n)$ be the set of all n-tuples over the first n

vertices of F not forming cycles: $B(0) = \varnothing$, $B(1) = \{((1_[, 1_[))\}$, $B(2) = \{((1_[, 1_[), (1_[, 1_[)), ((1_[, 1_[), (1_], 1_[)), ((1_[, 1_[), (1_[, 1_]))\}$,etc. Exact counting of $B(n)$ for all vertices of I is complex, so we use a different approach. We have a natural upper bound $|B(n)| \leq z^n$ with $z = 4$ when we consider all possible n-tuples.

Assign each $B(i)$ to one class: $C_1(i)$ contains all i-tuples of the form $(\ldots, (1_[, 1_[))$, and $C_2(i)$ contains all other i-tuples. We add every possible two-tuple to $C_1(i)$ and $C_2(i)$ to obtain two new classes $C_1(i+1)$ and $C_2(i+1)$ of $B(i+1)$. Adding two-tuple $(1_], 1_])$ to items from $C_1(i)$ is forbidden, as this directly gives us a cycle. Addition of $(1_[, 1_[)$ to i-tuples of both, $C_1(i)$ and $C_2(i)$ gives us $i+1$-tuples of class $C_1(i+1)$. Addition of $(1_[, 1_])$ or $(1_], 1_[)$ to either class leads to $i+1$-tuples of class $C_2(i+1)$, or might lead to infeasible encodings. Given these classes we create a 2×2 transition matrix A for the transposed vectors of the class cardinalities $(|C_1(i)|, |C_2(i)|)^T$ and $(|C_1(i+1)|, |C_2(i+1)|)^T$ such that $(|C_1(i+1)|, |C_2(i+1)|)^T \leq A\,(|C_1(i)|, |C_2(i)|)^T$. For large n we have that $(|C_1(n)|, |C_2(n)|)^T \leq A^n\,(|C_1(1)|, |C_2(1)|)^T \approx z^n x_1$ where z is largest real eigenvalue of A and x_1 is an eigenvector. Thus, z^n is a bound of $|B(n)|$. It follows that $A = \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}$. As the largest real eigenvalue of A is $2 + \sqrt{3}$, we have $z \leq 3.73205$ and bound $|B(n)| \leq 3.73205^n$.

Using these two classes eliminates all cycles over two consecutive vertices. By using three classes we can also prevent larger cycles and obtain tighter bounds for z:

- $C_1(i)$ contains all i-tuples $(\ldots, (1_[, 1_[), x)$, where x can consist of zero or more elements $(1_[, 1_])$, $(1_], 1_[)$ or $(1_], 1_[), (1_[, 1_])$ after each other.
- $C_2(i)$ contains all i-tuples $(\ldots, (1_[, 1_[), x, y)$ where y represents $(1_[, 1_])$ or $(1_], 1_[)$.
- $C_3(i)$ contains all other i-tuples.

Because we use three classes here, we can also prevents some cycles over more than two consecutive vertices. We obtain a 3×3 transition matrix A such that $(|C_1(i+1)|, |C_2(i+1)|, |C_3(i+1)|)^T \leq A\,(|C_1(i)|, |C_2(i)|, |C_3(i)|)^T$ of the form:

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 0 & 0 \\ 0 & 2 & 3 \end{pmatrix}$$

By calculating the largest real eigenvalue we obtain $z \leq 3.68133$. This bound is definitely not tight, it is possible to generalize this technique. We may take more classes into consideration, but already concerning two classes improves our results only incrementally. Computational research suggests that z is probably larger than 3.5. We replace 2^{2i} in Equation (3.6) by the last calculated value z^i to approximate the number of k-Planar Longest Path:

$$C(\ell) = O(\sum_{i=0}^{\frac{\ell}{2}} \binom{\frac{\ell}{2}}{i} 3^{\frac{\ell}{2}-i} 4^{\ell} z^{i}) = O(2^{3.37\ell}) \tag{3.8}$$

□

Using fast matrix multiplication

For k-Planar Longest Path, we can also apply the distance product technique for further runtime improvement.

Lemma 3.2.6. *k-Planar Longest Path on a planar graph G with branchwidth ℓ can be solved in time $O(2^{2.746\ell}\ell n + n^3)$.*

Proof. By Equation (3.8), we get

$$Z := (\frac{z}{2} + 1)^{\frac{\ell}{2}} \cdot 3^{\frac{\ell}{2}}$$

possible ways to update the states of F where $z \leq 3.68133$. That is, we obtain two matrices A, B, A with $4^{\frac{\ell}{2}}$ rows and Z columns and B with Z rows and $4^{\frac{\ell}{2}}$ columns. The rows of A and the columns of B represent the ways $L \cap O_L$ and $R \cap O_R$, respectively, contribute to the solution. The Z columns of A and rows B are ordered such that column i and row i form a valid assignment for all $1 \leq i \leq Z$ and we get

$$C := (-1) \cdot ([(-1) \cdot A] \star [(-1) \cdot B]),$$

the distance product of A and B with inverted signs for obtaining the maximum over the entries. Other than for previous distance product applications, we do not need to post-process matrix C, since the paths in Δ_L and Δ_R do not overlap.

By Theorem 2.2.5 in Section 2.2, we obtain an overall running time of $O(4^{(\omega-1)\frac{\ell}{2}} \cdot 6.68^{\frac{\ell}{2}} \cdot \ell n + n^3) = O(2^{2.746\ell}\ell n + n^3)$ for fast matrix multiplication constant $\omega < 2.376$. □

By Theorem 1.4.2 and Lemma 3.2.6 we achieve the running time $O(2^{10.984\sqrt{k}} n^{3/2} + n^3)$for k-Planar Longest Path. This completes the proof of Theorem 3.2.3.

3.2.2 Hamiltonian and Longest Path

One can apply the same techniques to obtain fast exact algorithms. The Planar Longest Path problem is solved by first computing an sc-decomposition of optimal width and than apply previous dynamic programming algorithm. The only difference is that having process the whole tree T of sc-decomposition (T, μ, π), we are interested at

value of the maximum entry of the root matrix rather than in a affirmative or rejecting answer concerning a parameter k.

For the weighted Planar Hamiltonian Path problem we there is some extra work. We are given a weighted Σ-plane graph G with weight function $w\colon E(G) \to N$ and we ask for a path of minimum weight through all vertices of $V(G)$. Again, we use the formulation via labeling: A labeling $\mathcal{H} : E(G) \to \{0,1\}$ is *Hamiltonian* if the subgraph $G_{\mathcal{H}}$ of G formed by the edges with label '1' is a spanning path. Find a Hamiltonian labeling $\mathcal{H}$ minimizing $\sum_{e\in E(G)} \mathcal{H}(e) \cdot w(e)$. For an edge labeling $\mathcal{H}$ and a vertex $v \in V(G)$ we define the *$\mathcal{H}$-degree* $deg_{\mathcal{H}}(v)$ of v as the sum of labels assigned to the edges incident to v. The algorithm though is almost identical to that of Planar Longest Path.

Only note, that if the weights become to large, we will not benefit from the matrix multiplication approach. With the same encoding as for k-Planar Longest Path, we restrict, in contrast, for every vertex $v \in F$ that $|c_L(v)| + |c_R(v)| = 2$ in order to prevent isolated vertices.Analysis is similar to that of Planar Longest Path, we get a slightly better running time since we do not have to account for isolated vertices.

From Proposition 1.4.9 we get a bound on the planar branchwidth and, thus, the following:

Theorem 3.2.7. Planar Longest Path (Cycle) *is solvable in time* $O(2^{5.827\sqrt{n}})$ Planar Hamiltonian Path (Cycle) *in time* $O(2^{5.579\sqrt{n}})$ *if the weights have size* $n^{O(1)}$ *otherwise in time* $O(2^{7.223\sqrt{n}})$ *and* $O(2^{6.903\sqrt{n}})$*, respectively.*

3.2.3 Planar Graph TSP

We consider an extended algorithm to the important problem Planar Graph TSP, mainly to give an insight on how far-reaching the techniques and structural results are, that we have introduced so far in the Section. Secondly, there are some nice technical details to it too, that are worth to be studied. In the Planar Graph TSP we are given a weighted Σ-plane graph G with weight function $w\colon E(G) \to N$ and we are asked for a shortest closed walk that visits all vertices of G at least once. Equivalently, this is TSP with distance metric the shortest path metric of G. We only sketch the algorithm for Planar Graph TSP since it is very similar to the algorithm for Planar Longest Path, and give a proof of the following:

Theorem 3.2.8. Planar Graph TSP *is solvable in time* $O(2^{8.15\sqrt{n}})$ *if the weights have size* $n^{O(1)}$ *otherwise in time* $O(2^{9.859\sqrt{n}})$.

Instead of collections of disjoint paths we now deal with connected components with even vertex degree for the vertices outside the nooses of the sc-decomposition. It is easy to show that a shortest closed walk passes through each edge at most twice. Thus every shortest closed walk in G corresponds to the minimum Eulerian subgraph in the graph G' obtained from G by adding to each edge a parallel edge. Every vertex of

an Eulerian graph is of even degree, which brings us to another equivalent formulation of the problem. A labeling $\mathcal{E} : E(G) \to \{0, 1, 2\}$ is *Eulerian* if the subgraph $G_{\mathcal{E}}$ of G formed by the edges with positive labels is a connected spanning subgraph and for every vertex $v \in V(G)$ the sum of labels assigned to edges incident to v is even. Thus Planar Graph TSP is equivalent to Minimum spanning Eulerian subgraph that is, to finding an Eulerian labeling $\mathcal{E}$ minimizing $\sum_{e \in E(G)} \mathcal{E}(e) \cdot w(e)$. For a labeling $\mathcal{E}$ and vertex $v \in V(G)$ we define the *$\mathcal{E}$-degree* $deg_{\mathcal{E}}(v)$ of v as the sum of labels assigned to the edges incident to v.

Let G be a Σ-plane graph and let (T, μ, π) be a rooted sc-decomposition of G of width ℓ. We use the same definitions for O_e, G_e, and Δ_e. We call a labeling $\mathcal{P}[e] \colon E(G_e) \to \{0, 1, 2\}$ a *partial Eulerian labeling* if the subgraph $G_{\mathcal{P}[e]}$ induced by the edges with positive labels satisfies the following properties:

- Every connected component of $G_{\mathcal{P}[e]}$ has a vertex in O_e.
- For every vertex $v \in V(G_e) \setminus O_e$, the $\mathcal{P}[e]$-degree $deg_{\mathcal{P}[e]}(v)$ of v is even and positive.

The weight of a partial Eulerian labeling $\mathcal{P}[e]$ is $\sum_{f \in E(G_e)} \mathcal{P}[e](f) \cdot w(f)$. Note that every partial Eulerian labeling of $G_{\{r,s\}}$ is also a Eulerian labeling.

As in Section 3.1, we define the set of all collections $G_{\mathcal{P}[e]}$ as $\mathbf{comp}_{G_e}(E(G_e), V(O_e))$ and because of sc-decompositions having Catalan structure, we get

$$\texttt{q-comp}_{G_e}(E(G_e), V(O_e)) = 4^{O(|V(O_e)|)}.$$

Again for obtaining a fast algorithm, we encode the information on which vertices of the connected components of $G_{\mathcal{P}[e]}$ of all possible partial Eulerian labelings $\mathcal{P}[e]$ hit $O_e \cap V(G)$. Also for every vertex $v \in O_e \cap V(G)$ the information if $deg_{\mathcal{P}[e]}(v)$ is either 0, or odd, or even and positive.

For a partial Eulerian labeling $\mathcal{P}[e]$ let C be a component of $G_{\mathcal{P}[e]}$ with at least two vertices in noose O_e. We scan the vertices of $V(C) \cap O_e$ according to the ordering π and mark with index '[' the first and with ']' the last vertex of C on O_e. We also mark by '$\square$' the other 'inner' vertices of $V(C) \cap O_e$. Finally we assign a numerical value.

If C has only one vertex in O_e, we mark this vertex by '0'. This includes the case $|V(C)| = 1$. Note that the connected components of $G_{\mathcal{P}[e]}$ form a non-crossing partition ncp. Thus, we can again decode the complete information on which vertices of each connected component of $V(G_{\mathcal{P}[e]})$ hit O_e.

Thus every such state must be an *algebraic term* with the indices '[' being the opening and ']' the closing bracket (with '$\square$' and '0' representing a possible term inside the brackets).

When encoding the parity of the vertex degrees, the following observation is useful: In every graph the number of vertices with odd degree is even. Consider a component C of $G_{\mathcal{P}[e]}$. There is an even number of vertices in $C \cap O_e$ with odd $\mathcal{P}[e]$-degree. Thus, we do not encode the parity of the degree of a vertex assigned by ']'. The parity is determined by the other vertices of the same component. The state of dynamic

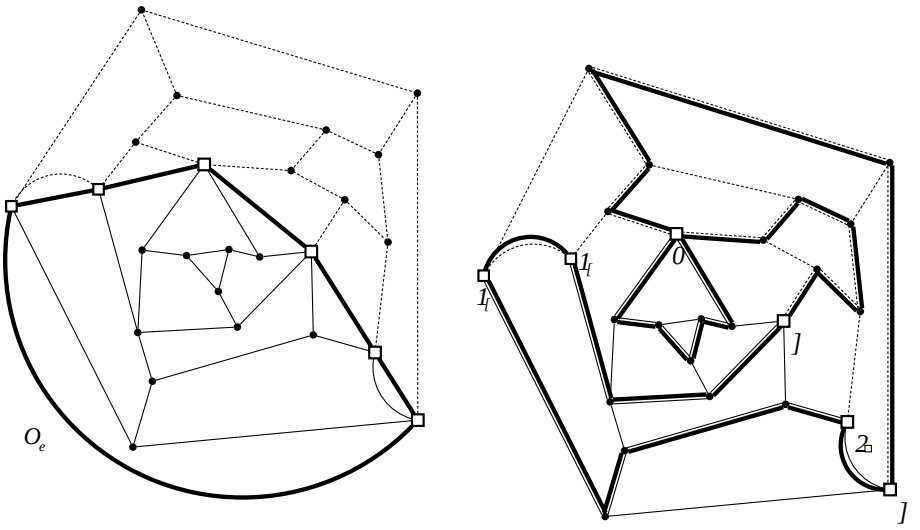


Figure 3.3: On the left we see a plane graph G— 3-connected and non-Hamiltonian— partitioned by the rectangle vertices hit by the marked noose O_e into G_e in drawn-through edges and $\overline{G_e}$ in dashed edges. To the right a subgraph $G_{\mathcal{E}}$ with Eulerian labeling $\mathcal{E}$ is marked. $G_{\mathcal{E}}$ is partitioned by the vertices of $O_e \cap V(G)$ which are labeled corresponding to partial Eulerian labeling $\mathcal{P}[e]$ of G_e. Encoding the vertices touched by O_e from the left to the right with $1_[, 1_[, 0,], 2_\square,]$, $G_{\mathcal{P}[e]}$ consists of three components C_1, C_2 and C_3 with $C_1 \cap O_e = \{1_[, 2_\square,]\}, C_2 \cap O_e = \{0\}, C_3 \cap O_e = \{1_[,]\}$. Here $G_{\mathcal{P}[e]}$ has edges only labeled with 1.

programming is $\vec{t_e} := (v_1, \ldots, v_\ell)$ with variables $v_1, \ldots, v_\ell$ having one of the six values: 0, $1_[$, $1_\square$, $2_[$, $2_\square$,]. Hence, there are at most $O(6^\ell |V(G)|)$ states. For every state, we compute a value $W_e(v_1, \ldots, v_\ell)$ that is the minimum weight over all partial Eulerian labelings $\mathcal{P}[e]$ encoded by $(v_1, \ldots, v_\ell)$:

- For every connected component C of $G_{\mathcal{P}[e]}$ with $|C \cap O_e| \geq 2$ the first vertex of $C \cap O_e$ in π is represented by $1_[$ or $2_[$ and the last vertex is represented by]. All other vertices of $C \cap O_e$ are represented by $1_\square$ or $2_\square$. For every vertex v marked by by $1_[$ or $1_\square$ the parity of $deg_{\mathcal{P}[e]}(v)$ is even and for every vertex v marked by by $2_[$ or $2_\square$, $deg_{\mathcal{P}[e]}(v)$ is positive and even.
- For every connected component C of $G_{\mathcal{P}[e]}$ with $v = C \cap O_e$, v is represented by 0. (Note that since for every $w \in V(G_e) \setminus O_e$ it holds that $deg_{\mathcal{P}[e]}(w)$ is even so must $deg_{\mathcal{P}[e]}(v)$.)
- Every vertex $v \in (V(G_e) \cap O_e) \setminus G_{\mathcal{P}[e]}$ is marked by 0. (Note that the vertices of the last two items can be treated in the same way in the dynamic programming.)

We put $W_e = +\infty$ if no such labeling exists. For an illustration of a partial Eulerian labeling see Figure 3.3. To compute an optimal Eulerian labeling we perform dynamic programming over middle sets as in the previous section. The first step of processing

the middle sets is to initialize the leaves corresponding to edges $f \in E$ of the graph G with values $W_e(0,0) = 0$, $W_e(1_[,]) = w(f)$, and $W_e(2_[,]) = 2w(f)$. Then, bottom-up, update every pair of states of two child edges e_L and e_R to a state of the parent edge e_P assigning a finite value W_P if the state corresponds to a feasible partial Eulerian labeling.

We compute all valid assignments to the variables $\vec{t_P} = (v_1, v_2, \ldots, v_p)$ from all possible valid assignments to the variables of $\vec{t_L}$ and $\vec{t_R}$. We define the numerical value $|\cdot|$ of ']' to be one if the sum of $deg_{\mathcal{P}[e]}$ over all vertices in the same component is odd, and to be two if the sum is even.

For every vertex $v \in (O_L \cup O_R \cup O_P) \cap V(G)$ we consider the three cases:

- $\underline{v \in \{L,R\}}$: $c_P(v) = c_L(v)$ if $v \in O_L \cap V(G)$, or $c_P(v) = c_R(v)$ otherwise.
- $\underline{v \in F}$: $(|c_L(v)| + |c_R(v)|) \, mod \, 2 = 0$ and $|c_L(v)| + |c_R(v)| > 0$.
- $\underline{v \in I}$: $|c_P(v)| = 0$ if $|c_L(v)|+|c_R(v)| = 0$, $|c_P(v)| = 1$ if $(|c_L(v)|+|c_R(v)|) \, mod \, 2 = 1$, else $|c_P(v)| = 2$.

Note that for a vertex $v \in O_P \cap V(G)$ it is possible that $|c_P(v)| = 0$ even if $|c_L(v)| + |c_R(v)|$ is even and positive since v might be the only intersection of a component with O_P. In order to verify that the encoding formed from two states of e_L and e_R corresponds to a labeling with each component touching O_P, we use an auxiliary graph A with $V(A) = (O_L \cup O_R) \cap V(G)$ and $\{v,w\} \in E(A)$ if v and w both are in one component of $G_{\mathcal{P}[e_L]}$ and $G_{\mathcal{P}[e_R]}$, respectively. Every component of A must have a vertex in $O_P \cap V(G)$. For every encoding of $\vec{t_P}$, we set $W_P = min\{W_P, W_L + W_R\}$.

For the root edge $\{r,s\}$ and its children e' and e'' note that $(O_{e'} \cup O_{e''}) \cap V(G) = F$ and $O_{\{r,s\}} = \varnothing$. Hence, for every $v \in V(G_{\mathcal{P}[\{r,s\}]})$ it must hold that $deg_{\mathcal{P}[\{r,s\}]}(v)$ is positive and even, and that the auxiliary graph A is connected. The optimal Eulerian labeling of G results from $min_{\vec{t}_{\{r,s\}}}\{W_r\}$.

Analyzing the algorithm, we obtain the following lemma.

Lemma 3.2.9. Planar Graph TSP *on a graph G with branchwidth at most ℓ can be solved in time $O(2^{3.84\ell}\ell n + n^3)$.*

Proof. Assume three adjacent edges e_P, e_L, and e_R of T with $|O_L| = |O_R| = |O_P| = \ell$ and that there are no intersection vertices. Thus we have $|F| = |L \cap O_L| = |R \cap O_R| = \frac{\ell}{2}$.

By just checking every combination of ℓ-tuples from O_L and O_R we obtain a bound of $O(\ell 6^{2\ell})$ for our algorithm.

This bound can be improved by using the fact that for all vertices $u \in F$ we want the sum of the assignments to be even, i.e., $(|c_L(u)| + |c_R(u)|) \, mod \, 2 = 0$.

We define $Q(\ell, m_1, m_2)$ as the number of ℓ-tuples over ℓ vertices of O_L and O_R, respectively, where the $\{0, 1_[, 1_\square, 2_[, 2_\square,]\}$ assignments for vertices from F is fixed and contains m_1 vertices of odd $\mathcal{P}[e]$-degree and m_2 vertices of even $\mathcal{P}[e]$-degree. The only

freedom is thus in the $\ell/2$ vertices in $L \cap O_L$ and $R \cap O_R$, respectively:

$$\begin{aligned} Q(\ell, m_1, m_2) &= O(\sum_{i=0}^{\frac{\ell}{2}} \sum_{j=0}^{\frac{\ell}{2}-i} \binom{\frac{\ell}{2}}{i} \binom{\frac{\ell}{2}-i}{j} 1^{\frac{\ell}{2}-i-j} (\frac{5}{2})^{i+m_1} (\frac{5}{2})^{j+m_2}) \\ &= O(6^{\frac{\ell}{2}} (\frac{5}{2})^{m_1} (\frac{5}{2})^{m_2}) \end{aligned} \tag{3.9}$$

This expression is a summation over the number of vertices of odd and even $\mathcal{P}[e]$-degree in $L \cap O_L$ and $R \cap O_R$, respectively. The terms $\binom{\frac{\ell}{2}}{i}$ and $\binom{\frac{\ell}{2}-i}{j}$ count the possible locations for the vertices of odd and even $\mathcal{P}[e]$-degree, respectively, whereas $(\frac{5}{2})^{i+m_1}$ and $(\frac{5}{2})^{i+m_2}$ count the number of those assignments. The $1^{\frac{\ell}{2}-i-j}$ is left in the formula to represent the assignment of $\mathcal{P}[e]$-degree zero to the remaining $\ell/2 - i - j$ vertices.

We define $C(\ell)$ as the number of possibilities of forming an ℓ-tuple from O_P. We sum over i and j: the number of vertices of odd and even $\mathcal{P}[e]$-degree in the assignment for F:

$$C(\ell) = \sum_{i=0}^{\frac{\ell}{2}} \sum_{j=0}^{\frac{\ell}{2}-i} \binom{\frac{\ell}{2}}{i} \binom{\frac{\ell}{2}-i}{j} 5^{\frac{\ell}{2}-i-j} Q(\ell, i, j)^2 \tag{3.10}$$

The term $5^{\frac{\ell}{2}-i}$ denotes the number of ways the vertices of F can be assigned from one side with $\mathcal{P}[e]$-degree zero and from the other side with even $\mathcal{P}[e]$-degree. Straightforward calculation yields:

$$\begin{aligned} C(\ell) &= O(\sum_{i=0}^{\frac{\ell}{2}} \sum_{j=0}^{i} \binom{\frac{\ell}{2}}{i} \binom{\frac{\ell}{2}-i}{j} 5^{\frac{\ell}{2}-i-j} 6^{\ell} (\frac{5}{2})^{2i} (\frac{5}{2})^{2j}) \\ &= O((6\sqrt{17.5})^{\ell}) \end{aligned} \tag{3.11}$$

We obtain an overall running time of $O(6^{\ell}(\frac{35}{2})^{\frac{\ell}{2}}\ell n)$.

Again we can further improve upon the previous bound by only forming encodings that do not create several components. In contrast to cycles, the components can be formed at the vertices in F with numerical part 1 and 2 in both O_L and O_R. But we only consider vertices with even sum of the numerical part of the assignment. Thus, we look separately at the classes of even $\mathcal{P}[e]$-degree and odd $\mathcal{P}[e]$-degree. Without loss of generality consider odd $\mathcal{P}[e]$-degree: as in the previous section we want to exclude the case $((1_[, 1_[), (],]))$. Again suppose the two classes: $C_1(i)$ contains all i-tuples $(\ldots, (1_[, 1_[))$, and $C_2(i)$ contains all other i-tuples. Adding $(],])$ to i-tuples from $C_1(i)$ is forbidden, as this will lead to a single component. Addition of $(1_[, 1_[)$ to i-tuples of both, $C_1(i)$ and $C_2(i)$ gives us the $i+1$-tuples of class $C_1(i+1)$. We obtain the matrix $A = \begin{pmatrix} 1 & 1 \\ 5 & \frac{21}{4} \end{pmatrix}$

with largest eigenvalue $z = \frac{\sqrt{77}+9}{2} \leq 6.2097$. We can insert z for both, the odd and the even valued vertices separately in Equation (3.11):

$$\begin{aligned} C(\ell) &= O(\sum_{i=0}^{\frac{\ell}{2}} \sum_{j=0}^{i} \binom{\frac{\ell}{2}}{i} \binom{\frac{\ell}{2}-i}{j} 5^{\frac{\ell}{2}-i-j} 6^{\ell} z^{i} z^{j}) \\ &= O((6\sqrt{17.4195})^{\ell}) \end{aligned} \tag{3.12}$$

Using distance product, we improve for small weights the bound to

$$C(\ell) = O((6^{\frac{\omega-1}{2}}\sqrt{17.4195})^{\ell}).$$

□

This completes the proof of Theorem 3.2.8.

3.2.4 Variants

In this section we will discuss results on other edge-subset problems on planar graphs.

Minimum Number Cycle Cover.

Minimum Number (Cost) Cycle Cover asks for a minimum number (cost) of vertex disjoint cycles that cover the vertex set of the input graph. The algorithm can be implemented as a variant of Planar Hamiltonian Cycle algorithm, with the additional freedom of allowing cycles in the merging step. Thus the result from Equation (3.7) can be used directly, leading to a running time of $O(n^{\frac{3}{2}}2^{5.663\sqrt{n}})$ for weights of size $n^{O(1)}$.

Problems with tree-like solutions.

The problem Connected Dominating Set asks for a minimum Dominating Set that induces a connected subgraph. See [35] for a subexponential algorithm on graphs of bounded outerplanarity. Connected Dominating Set can be formulated as Max Leaf Problem where one asks for a spanning tree with the maximum number of leaves.

For the state of the vertices on the nooses we can use an encoding with symbols $0_0, 0_1, 1_0, 1_{[}, 1_{\square}, 1_{]}$. The numerical part indicates whether (1) or not (0) a vertex is an inner node of the solution spanning tree. The indices for the vertices labeled with a 1 encode to which connected component they belong, 1_0 is an isolated vertex that becomes an inner node. The indices for the leaves 0 indicate if a vertex is connected (1) or not (0) to any vertex marked as an inner node. Using our technique, we obtain for Connected Dominating Set a running time of $O(2^{8.111\sqrt{n}})$ for weights of size $n^{O(1)}$.

Table 3.1: Worst-case runtime in the upper part expressed also by treewidth **tw** and branchwidth **bw** of the input graph. The lower part gives a summary of the most important improvements on exact and parameterized algorithms with parameter k and compares the results with and without applying fast matrix multiplication for unweighted and weighted problems, respectively. Note that we use the fast matrix multiplication constant $\omega < 2.376$.

	Previous results	New results
w Planar Hamiltonian Cycle	—	$O(n2^{\min\{2.58\,\mathbf{tw},3.31\,\mathbf{bw}\}})$
uw Planar Hamiltonian Cycle	—	$O(n2^{\min\{2.58\,\mathbf{tw},2.66\,\mathbf{bw}\}})$
w/uw Planar Hamiltonian Path	$O(2^{2.29\sqrt{n}\log n})$ [50]	$O(2^{7.2\sqrt{n}})/O(2^{5.83\sqrt{n}})$
w/uw Planar Graph TSP	$2^{O(\sqrt{n}\log n)}$ [72]	$O(2^{9.86\sqrt{n}})/O(2^{8.15\sqrt{n}})$
w/uw Planar connected DS	$2^{O(\sqrt{n}\log n)}$ [72]	$O(2^{9.82\sqrt{n}})/O(2^{8.11\sqrt{n}})$
w/uw Planar Steiner Tree	$2^{O(\sqrt{n}\log n)}$ [72]	$O(2^{8.49\sqrt{n}})/O(2^{7.16\sqrt{n}})$
w/uw Plan Feedback Vertex Set	$2^{O(\sqrt{n}\log n)}$ [72]	$O(2^{9.26\sqrt{n}})/O(2^{7.56\sqrt{n}})$
Param Planar Longest Path	—	$O(2^{10.5\sqrt{k}}k + n^3)$

The Steiner Tree of some subset X of the vertices of a planar graph G is a minimum-weight connected subgraph of G that includes X. It is always a tree; thus, we only encode connected subgraphs by using four symbols $0, [,], \square$. Here, $[,], \square$ mark the first, the last and all other vertices of a component and 0 marks isolated vertices and vertices that are the only intersection of a component and the noose. Note that every vertex of X must be part of a component, whereas the vertices of $V(G) \setminus X$ must not. We obtain for Steiner Tree a running time of $O(2^{7.163872\sqrt{n}})$ for weights of size $n^{O(1)}$ applying distance product.

In Feedback Vertex Set on an undirected planar graph G, one is asked to find a set Y of vertices of minimum cardinality such that every cycle of G passes through at least one vertex of Y. Feedback Vertex Set is equivalent to the problem: find an *induced* forest F in G with vertex set $V(F)$ of maximum cardinality. It holds that $V(G) \setminus Y = V(F)$. We are able to encode induced connected subgraphs with our technique. We mark if a vertex is in $V(F)$ or not. Every edge of G is an edge in the forest if its incident vertices are in $V(F)$. We can solve Feedback Vertex Set in time $O(2^{9.264\sqrt{n}})$.

In the parameterized version of the problem, k-Feedback Vertex Set, we ask if Y is of size at most parameter k. We improve the $2^{O(\sqrt{k}\log k)}n^{O(1)}$ algorithm in [67] to $2^{O(\sqrt{k})}n^{O(1)}$ by using the *bidimensionality* of k-Feedback Vertex Set (see [31] for more information). If a problem on graphs of bounded treewidth **tw** is solvable in time $2^{O(\mathbf{tw})}n^{O(1)}$ and its parameterized version with parameter k is bidimensional then it is solvable in time $2^{O(\sqrt{k})}n^{O(1)}$.

We summarize the results in Table 3.1.

3.3 Torus-embedded graphs

This section will extend the dynamic programming algorithm for Planar Longest Path of Section 3.2 to the Hamiltonian Cycle problem on graphs embedded on a torus. Since the resulting constants are pretty small, we give a detailed description on how to use an efficient encoding for fast computation. We thereby reduce the torus-related problem A to a collection of planar-graph problems $A_1, \ldots, A_q$, where A has a solution if and only if there is a solution for one of $A_1, \ldots, A_q$. The torus case includes the main ideas for extending to surfaces of higher genus. In Section 3.4 we will not give an explicit encoding, since the constant are becoming too large. As part of the algorithm for H-minor-free graphs, we give in Section 3.5 a parametrized algorithm for the k-Longest Path problem on graph of bounded genus.

3.3.1 Hamiltonian Cycle

The idea behind solving the Hamiltonian Cycle problem on $\mathbb{S}_1$-embedded graphs is to suitably modify the graph G in such a way that the new graph G' is $\mathbb{S}_0$-embedded (i.e. planar) and restate the problem to an equivalent problem on G' that can be solved by dynamic programming on a sc-decomposition of G'.

Planarization.

Let G be an $\mathbb{S}_1$-embedded graph (i.e. a graph embedded on the torus). By Proposition 1.2.3, it is possible to find in polynomial time a shortest noncontractible (tight) noose N of G. Let G' be the graph obtained by cutting along N on G. By Proposition 1.2.5, G' is $\mathbb{S}_0$-embeddable.

Definition 3.3.1. *A* cut *of a Hamiltonian cycle C in G along a tight noose N is the set of disjoint paths in G' resulting by cutting G along N.*

Each cut-noose N_X and N_Y borders an open disk Δ_X and Δ_Y, respectively, with $\Delta_X \cup \Delta_Y = \varnothing$. Let $x_i \in N_X$ and $y_i \in N_Y$ be duplicated vertices of the same vertex in N.

The following definition is very similar, in fact a restriction of the definition of **paths** in Section 3.1.

Definition 3.3.2. *A set of disjoint paths $\mathbf{P}$ in G' is* relaxed Hamiltonian *if:*

(P1) Every path has its endpoints in N_X and N_Y.

(P2) Vertex x_i is an endpoint of some path P if and only if y_i is an endpoint of a path $P' \neq P$, unless $|\mathbf{P}| = 1$.

(P3) For x_i and y_i: one is an inner vertex of a path if and only if the other is not in any path.

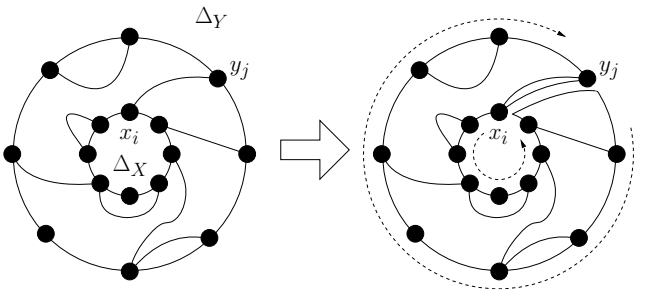


Figure 3.4: **Cut-nooses.** In the left diagram, one equivalence class of relaxed Hamiltonian sets is illustrated. All paths have endpoints in N_X and N_Y. Fix one path with endpoints x_i and y_i. In the right diagram we create a tunnel along this path. The empty disks Δ_X and Δ_Y are united to a single empty disk. Thus, we can order the vertices bordering the disk to π_{XY}.

(P4) Every vertex of $G' \setminus (N_X \cup N_Y)$ *is in some path.*

For the case $|\mathbf{P}| = 1$, $\mathbf{P}$ might as well consist of a single cycle. A *cut* of a Hamiltonian cycle in G is a relaxed Hamiltonian set in G', but not every relaxed Hamiltonian set in G' forms a Hamiltonian cycle in G. However, given a relaxed Hamiltonian set $\mathbf{P}$ one can check in linear time (by identifying the corresponding vertices of N_X and N_Y) if $\mathbf{P}$ is a cut of a Hamiltonian cycle in G.

Define the equivalence relation $\sim$ on the set $\mathbf{HS}(G')$ of all relaxed Hamiltonian sets in G': for any two sets $\mathbf{P}_1, \mathbf{P}_2 \in \mathbf{HS}(G')$, $\mathbf{P}_1 \sim \mathbf{P}_2$ if for every path in $\mathbf{P}_i$ there is a path in $\mathbf{P}_j$ with the same endpoints $(1 \leq i, j \leq 2)$. Denote by $\mathbf{HS}(G')/\sim$ the quotient set of $\mathbf{HS}(G')$ by $\sim$. Note that $\mathbf{HS}(G')$ without property $(P3)$ is a subset of $\mathbf{paths}_{G'}(E(G'), N_X \cup N_Y)$.

Lemma 3.3.3. *Let* G' *be a* $\mathbb{S}_0$*-embedded graph obtained from a* $\mathbb{S}_1$*-embedded graph* G *by cutting along a tight noose* N*. Then* $|\mathbf{HS}(G')/\sim|$ *is* $O(k^2\, 2^{3k})$*, where* k *is the length of* N*.*

Proof. By Lemma 3.1.4 in Section 3.1, we have that $\texttt{q-paths}_{G'}(E(G'), N_X \cup N_Y) = O(k^2\, 2^{2k})$. That is, we guess two vertices $x_i \in N_X$ and $y_j \in N_Y$ being two fixed endpoints of a path $P_{i,j}$ in a relaxed Hamiltonian set $\mathbf{P}$. We 'cut' the sphere $\mathbb{S}_0$ along $P_{i,j}$ and, that way, create a "tunnel" between Δ_X and Δ_Y unifying them to a single disk Δ_{XY}. Take the counter-clockwise order of the vertices of N_X beginning with x_i and concatenate N_Y in clockwise order with y_j the last vertex. We denote the new cyclic order by π_{XY} (see Figure 3.4 for an example) .

Note that we also include the case of those different equivalent sets of paths, where N_X and N_Y are not connected by any path. That is, each path has both endpoints in either only N_X or only N_Y. We now count $\mathbf{HS}(G')/\sim$. Apparently, in a feasible solution, if a vertex $x_h \in N_X$ is an inner vertex of a path, then $y_h \in N_Y$ does not belong to any path and vice versa. With (P3), there are two more possibilities for the pair of

vertices x_h, y_h to correlate with a path. With $|N_X| = |N_Y| = k$, the overall upper bound the quotient set of $\mathbf{HS}(G')$ is $O(k^2\, 2^{3k})$. □

We call a *candidate* $\mathbf{C}$ of the quotient set $\mathbf{HS}(G')/\sim$ to be a set of arcs with vertices only in $N_X \cup N_Y$ representing a set of equivalent relaxed Hamiltonian sets. Thus for each candidate we fix a path between N_X and N_Y and define the order π_{XY}. By making use of dynamic programming on sc-decompositions we check for each candidate $\mathbf{C}$ if there is a spanning subgraph of the planar graph G' isomorphic to a relaxed Hamiltonian set $\mathbf{P} \sim \mathbf{C}$.

The algorithm

Instead of looking at the Hamiltonian cycle problem on G we solve the relaxed Hamiltonian set problem on the $\mathbb{S}_0$-embedded graph G' obtained from G: Given a candidate $\mathbf{C}$, i.e. a set of vertex tuples $\mathbf{T} = \{(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\}$ with $s_i, t_i \in N_X \cup N_Y, i = 1, \ldots, k$ and a vertex set $\mathbf{I} \subset N_X \cup N_Y$. Does there exist a relaxed Hamiltonian set $\mathbf{P}$ such that every (s_i, t_i) marks the endpoints of a path and the vertices of $\mathbf{I}$ are inner vertices of some paths?

Our algorithm works as follows: first encode the vertices of $N_X \cup N_Y$ according to $\mathbf{C}$ by making use of the Catalan structure of $\mathbf{C}$ as it follows from the proof of Lemma 3.3.3. We may encode the vertices s_i as the 'beginning' and t_i as the 'ending' of a path of $\mathbf{C}$. Using order π_{XY}, we ensure that the beginning is always connected to the next free ending. This allows us to design a dynamic programming algorithm using a small constant number of states. We call the encoding of the vertices of $N_X \cup N_Y$ *base encoding* to differ from the encoding of the sets of disjoint paths in the graph. We proceed with dynamic programming over middle sets of a rooted sc-decomposition (T, μ, π) in order to check whether G' contains a relaxed Hamiltonian set $\mathbf{P}$ equivalent to candidate $\mathbf{C}$. As T is a rooted tree, this defines an orientation of its edges toward its root. Let e be an edge of T and let O_e be the corresponding tight noose in $\mathbb{S}_0$. Recall that the tight noose O_e partitions $\mathbb{S}_0$ into two discs which, in turn, induces a partition of the edges of G into two sets. We define as G_e the graph induced by the edge set that corresponds to the "lower side" of e it its orientation toward the root. All paths of $\mathbf{P} \cap G_e$ start and end in O_e and $G_e \cap (N_X \cup N_Y)$. For each G_e, we encode the equivalence classes of sets of disjoint paths with endpoints in O_e. From the leaves to the root for a parent edge and its two children, we update the encodings of the parent middle set with those of the children as described in Section 2.1. We obtain the algorithm in Figure 3.5.

Lemma 3.3.4. *For a given a sc-decomposition (T, μ, π) of G' of width ℓ and a candidate $\mathbf{C} = (\mathbf{T}, \mathbf{I})$ the running time of one* **For***-loop in the main step of* **HamilTor** *on $\mathbf{C}$ is $O(2^{6.36\ell} \cdot |V(G')|^{O(1)})$.*

In the proof of the lemma we show how to apply the dynamic programming step of **HamilTor**. We sketch only the main idea here and give the detailed algorithm in the appendix of this chapter (Section 3.6).

Algorithm **HamilTor**
Input: $\mathbb{S}_1$-embedded graph G.
Output: Decision/Construction of the HAMILTONIAN CYCLE problem on G.

Preliminary Step: Cut G along a shortest noncontractible (tight) noose N and
output the $\mathbb{S}_0$-embedded graph G' and the cut-nooses N_X,N_Y in G'.
Main step: **For** all candidates $\mathbf{C} \in \mathbf{HS}(G')/\sim$ {
If C is equivalent to a Hamiltonian cycle in G
when identifying the duplicated vertices in N_X,N_Y in G' {
Determine the pair of endpoints (s,t) that build the first and last vertex in π_{XY}.
Generate a base encoding of the vertices of N_X and N_Y, representing the candidate **C**.
Compute a rooted sc-decomposition (T,μ,π) of G'.
From the leaves to the root on each middle set O_e of T bordering G_e {
Do dynamic programming —
find all path collections of $\mathbf{paths}_{G_e}(E(G_e), V(G_e) \cap [O_e \cup (N_X \cup N_Y)])/\sim$
with respect to the base encoding of N_X,N_Y.}
If there exists a $\mathbf{P} \in \mathbf{paths}_{G'}(E(G'), N_X \cup N_Y)/\sim$ with $\mathbf{P} \sim \mathbf{C}$, then {
Reconstruct **P** from the root to the leaves of T and
output corresponding Hamiltonian cycle.} } }
Output "No Hamiltonian Cycle exists".

Figure 3.5: Algorithm **HamilTor**.

For a dynamic programming step we need the information on how a tight noose O_e and $N_X \cup N_Y$ intersect in G' and which parts of $N_X \cup N_Y$ are a subset of the subgraph G_e. Define the set of O-arcs $\mathcal{X} = (G_e \setminus O_e) \cap (N_X \cup N_Y)$. G_e is bordered by O_e and $\mathcal{X}$ and thus partitioned into several edge-disjoint components that we call *partial components*. Each partial component is bordered by a noose that is the union of subsets of O_e and $\mathcal{X}$. Let us remark that this noose is not necessarily tight. The partial components intersect pairwise only in vertices of $\mathcal{X} \cap O_e$ that we shall define as *connectors* (See Figure 3.10 in Section 3.6). In each partial component we encode a collection of paths with endpoints in the bordering noose using Catalan structures. The union of these collections over all partial components must form a collection of paths in G_e with endpoints in O_e and in $\mathcal{X}$. We ensure that the encoding of the connectors of each two components fit. During the dynamic programming we need to keep track of the base encoding of $\mathcal{X}$. We do so by only encoding the vertices of O_e without explicitly memorizing with which vertices of $\mathcal{X}$ they form a path. With several technical tricks we can encode O_e such that two paths with an endpoint in O_e and the other in $\mathcal{X}$ can be connected to a path of **P** only if both endpoints in $\mathcal{X}$ are the endpoints of a common path in **C**.

Runtime analysis

To finish the estimation of the running time we need some combinatorial results.

Lemma 3.3.5. *Let G be a $\mathbb{S}_1$-embedded graph on n vertices and G' the planar graph obtained by cutting along a noncontractible tight noose of G. Then* $\mathbf{bw}(G') \leq \sqrt{4.5} \cdot \sqrt{n} + 2$.

Proof. Let N_X and N_Y be the cut-nooses in G' bordering the empty disks Δ_X and Δ_Y. We will prove the lemma assuming that after cutting along a noncontractible tight noose N of G, all edges with both ends in N are incident to N_X (the general case is a slightly more technical implementation of the same idea). We construct a new graph G^* by removing the vertices of N_Y from G'. Thus $|V(G^*)| = |V(G)| = n$. By Proposition 1.4.9 in Section 1.2, there is a sc-decomposition (T, μ, π) of G^* of width at most $\sqrt{4.5} \cdot \sqrt{n}$. Δ_Y is part of a region R of G^* bordered by a closed walk C. The neighborhood of N_Y in G' is a subset of the vertices of C in G^*. Let E_Y be the set of edges in G' incident to N_Y. Note that $E(G^*) \cup E_Y = E(G')$ and that E_Y induces a graph that is a subgraph that can be seen as a union of stars whose centers lay on C (this is based on the assumption that no edge has both ends in N_Y). We construct a branch-decomposition of G' from (T, μ, π) by doing the following. For every edge $x \in E_Y$ we choose edge y of C having a common endpoint v with x and being the next edge of C in counter-clockwise order incident to v. Let e_y be the edge of T adjacent to the leaf of corresponding to y. We subdivide e_y by placing a new vertex on it and attach a new leaf corresponding to x. We claim that the width of the new branch-decomposition (T', μ') is at most the width of (T, μ, π) plus two. For an edge y of C we may subdivide e_y of T several times creating a subtree T_y. But all the middle sets of the edges T_y have only one vertex in common, namely the common endpoint v. The middle set connecting T_y to T may have up to two more vertices that are, in order of appearance in π_{XY}, the first and the last endpoints of the considered edges of E_Y. Let $E(C)$ be the edge set of C. Since (T, μ, π) is a sc-decomposition, we have that for every e of $E(T)$, if the corresponding tight noose O_e bordering G_e^* intersects region R bordered by C, then $E(C) \cap G_e^*$ induces a connected subset of C. Note that in contrast O_e and C may intersect in single vertices only. Thus, O_e and that subset intersect in only two vertices v, w. v and w each have at most one adjacent vertex in N_Y that is connected to $C \setminus G_e^*$. Hence each middle set of T' has at most two vertices more than the corresponding middle set of T. □

Lemma 3.3.6. *Let G be a $\mathbb{S}_1$-embedded graph on n vertices. Then* $\mathbf{rep}(G) \leq \sqrt{4.5} \cdot \sqrt{n} + 2$.

Proof. Let G' be the $\mathbb{S}_0$-embedded graph obtained by cutting along a noncontractible tight noose N of G. By Lemma 3.3.5, there is a sc-decomposition (T, μ, π) of G' of width at most $\sqrt{4.5} \cdot \sqrt{n} + 2$. We subdivide an arbitrary edge e of T into the edges e_1,e_2 and root the tree at the new node r. Assume that for one of e_1, e_2, say e_1, both cut-nooses N_X and N_Y are properly contained in G_{e_1}. We traverse the tree from e_1 to the leaves. We always branch toward a child edge e with middle set O_e such that $N_X \cup N_Y \subset G_e$. At some point we reach an e with either *a*) G_e properly containing exactly one cut-noose or *b*) O_e intersecting both cut-nooses or *c*) G_e properly containing one cut-noose and O_e intersecting the other. In case *c*) we continue traversing from e toward the leafs always branching toward the edge with *c*) until we reach an edge with either *a*) or *b*). In case *a*), tight noose O_e forms a noncontractible tight noose in G, hence the length of O_e must be at least the representativity of G. In case *b*), O_e is the union of two lines with the shortest, say N_1, of length at most $\frac{|O_e|}{2}$. But both endpoints of N_1 are connected in the

$\mathbb{S}_1$-embedded graph G by a line N_2 of N of length L with $0 \leq L \leq \frac{|N|}{2}$. N_1 and N_2 form a noncontractible tight noose in G of length at most $\frac{|O_e|}{2} + \frac{|N|}{2}$. Hence, $|O_e| \geq \mathbf{rep}(G)$. □

Putting all together we obtain the following theorem.

Theorem 3.3.7. *Let G be a graph on n vertices embedded on a torus $\mathbb{S}_1$. The* Hamiltonian cycle *problem on G can be solved in time $O(2^{19.856\sqrt{n}} \cdot n^{O(1)})$.*

Proof. We run the algorithm **HamilTor** on G. The algorithm terminates positively when the dynamic programming is successful for some candidate of an equivalence class of relaxed Hamiltonian sets and this candidate is a cut of a Hamiltonian cycle. By Proposition 1.2.3, the Preliminary Step can be performed in polynomial time. Let k be the minimum length of a noncontractible noose N, and let G' be the graph obtained from G by cutting along N. By Lemma 3.3.3, the number of all candidates of relaxed Hamiltonian sets in G' is $O(2^{3k}) \cdot n^{O(1)}$. So the main step of the algorithm is called $O(2^{3k}) \cdot n^{O(1)}$ times. By Proposition 2.3.1, an optimal branch-decomposition of G' of width ℓ can be constructed in polynomial time. By Lemma 3.3.4, dynamic programming takes time $O(2^{6.36\ell}) \cdot n^{O(1)}$. Thus the total running time of **HamilTor** is $O(2^{6.36\ell} \cdot 2^{3k}) \cdot n^{O(1)}$. By Lemma 3.3.6, $k \leq \sqrt{4.5} \cdot \sqrt{n} + 2$ and by Lemma 3.3.5, $\ell \leq \sqrt{4.5} \cdot \sqrt{n} + 2$, and the theorem follows. □

3.4 Graphs embedded on surfaces of bounded genus

Now we extend our algorithm to graphs of higher genus. For this, we use the following kind of planarization: We apply Proposition 1.2.5 and cut iteratively along shortest noncontractible nooses until we obtain a planar graph G'. If at some step G' is the disjoint union of two graphs G_1 and G_2, we apply Proposition 1.2.5 on G_1 and G_2 separately.

We examine how a shortest noncontractible noose affects the cut-nooses of previous cuts:

Definition 3.4.1. *Let $\mathcal{K}$ be a family of cycles in G. We say that $\mathcal{K}$ satisfies the* 3-path-condition *if it has the following property. If x, y are vertices of G and P_1, P_2, P_3 are internally disjoint paths joining x and y, and if two of the three cycles $C_{i,j} = P_i \cup P_j$, $(1 \leq i < j \leq 3)$ are not in $\mathcal{K}$, then also the third cycle is not in $\mathcal{K}$.*

Proposition 3.4.2. *(Mohar and Thomassen [74]) The family of Σ-noncontractible cycles of a Σ-embedded graph G satisfies the 3-path-condition.*

Proposition 3.4.2 is useful to restrict the number of ways not only on how a shortest noncontractible tight noose may intersect a face but as well on how it may intersect the vertices incident to a face.

Lemma 3.4.3. *Let G be Σ-embedded and F a face of G bordered by $V_1 \subseteq V(G)$. Let $\overline{F} := V_1 \cup F$. Let N_s be a shortest noncontractible (tight) noose of G. Then one of the following holds*

1) $N_s \cap \overline{F} = \varnothing$.
2.1) $N_s \cap F = \varnothing$ *and* $|N_s \cap V_1| = 1$.
2.2) $N_s \cap F = \varnothing, N_s \cap V_1 = \{x, y\}$, *and x and y are both incident to one more face different than F which is intersected by N_s.*
3) $N_s \cap F \neq \varnothing$ *and* $|N_s \cap V_1| = 2$.

Proof. Recall that N_s is tight i.e. it can be seen as a cycle in the radial graph R_G. This directly implies that if $|N_s \cap V_1| = 1$, then $N_s \cap F = \varnothing$.

Suppose now that $N_s \cap V_1 = \{v_i, v_j\}$ and $N_s \cap F = \varnothing$. Suppose also that there is no face as the one required in 2.2. Then the cycle C_s of R_G corresponding to N_s is partitioned into two paths P_2 and P_3, each with ends v_i and v_j and of length > 2. We use the notation v_F for the vertex of R_G corresponding to the face F. Let also $P_1 = (v_i, v_F, v_j)$ and notice that the two cycles of R_G defined by $P_1 \cup P_3$ and $P_1 \cup P_2$ have length smaller than $P_2 \cup P_3 = C_s$ and therefore they are contractible. By Proposition 3.4.2, N_s is contractible—a contradiction.

For the sake of contradiction, we assume that $|N_s \cap V_1| \geq 3$. Assume N_s intersects V_1 in vertices $I = v_1, \ldots, v_k$, $k \geq 3$, and with at most two vertices connected by the part of the noose of N_s that intersects F. In the radial graph R_G of G, N_s corresponds to the shortest noncontractible cycle C_s. In R_G each vertex of V_1 is a neighbor of the vertex v_F.

We consider the two cases: $N_s \cap F \neq \varnothing$. That is, there exists a path $\{v_i, v_F, v_j\} \subset C_s$ in R_G with $v_i, v_j \in I$. Let v_h be another vertex in $I = V_1 \cap C_s$. Consider the three paths in R_G connecting v_F and v_h, namely $P_1 = (v_F, v_i, \ldots, v_h)$, $P_2 = (v_F, v_j, \ldots, v_h)$, and $P_3 = (v_F, v_h)$. Notice also that the two cycles of R_G defined by $P_1 \cup P_3$ and $P_2 \cup P_3$ have length smaller than $P_1 \cup P_2 = C_s$ and therefore they are contractible which is a contradiction to Proposition 3.4.2.

In case $N_s \cap F = \varnothing$, we choose $v_i, v_j, v_h \in I$ arbitrarily and the arguments of the previous case imply that the path $P_1 \cup P_2$ is contractible. We define now the paths $Q_1 = (v_i, \ldots, v_h, \ldots, v_j)$, $Q_2 = (v_i, v_F, v_j)$, and $Q_3 = (v_i, \ldots, v_j)$ between the vertices v_i and v_j. As $Q_1 \cup Q_2 = P_1 \cup P_2$, the cycle $Q_1 \cup Q_2$ of R_G is contractible. The same holds for the cycle $Q_2 \cup Q_3$ as its length is less than the length of $Q_1 \cup Q_3 = C_s$. Then again Proposition 3.4.2 implies that $Q_1 \cup Q_3 = C_s$ is contractible, a contradiction.

□

Lemma 3.4.4. *Given a Σ-embedded graph G where $\Sigma \neq \mathbb{S}_0$, with n vertices. Let $G^{(i)}$ be the $\Sigma^{(i)}$-embedded graph obtained by iteratively cutting G i times along a shortest noncontractible noose. Then* $\mathbf{rep}(G^{(i)}) \leq (\sqrt{4.5} + 2 \cdot \sqrt{2 \cdot \mathbf{eg}(\Sigma)})\sqrt{n}$.

Proof. Given graph $G^{(i)}$ embedded in surface $\Sigma^{(i)}$. By Proposition 1.2.5, $\mathbf{eg}(\Sigma^{(i)}) \leq \mathbf{eg}(\Sigma) - i$. We obtain graph $G^{(i)}_{del}$ by removing all duplicated vertices from $G^{(i)}$ that have been introduced by cutting along the nooses $N^{(0)}, \ldots, N^{(i-1)}$. Since $|V(G^{(i)}_{del})| = |V(G)|$ we have with Proposition 1.2.4 that $\mathbf{rep}(G^{(i)}_{del}) \leq \mathbf{bw}(G^{(i)}_{del}) \leq (\sqrt{4.5}+2\cdot\sqrt{2\cdot\mathbf{eg}(\Sigma^{(i)})})\sqrt{n}$. Let $N^{(i)}_{del}$ be a shortest noncontractible noose of $G^{(i)}_{del}$ that we consider. Since $N^{(i)}_{del}$ is tight and there are at most $i+1$ faces in $G^{(i)}_{del}$ that formerly contained the removed duplicated vertices of $G^{(i)}$, we obtain with Lemma 3.4.3 a noncontractible noose $N^{(i)}$ in $G^{(i)}$ consisting of $N^{(i)}_{del}$ and at most $2(i+1)$ duplicated vertices of $N^{(0)}, \ldots, N^{(i-1)}$. Hence, $\mathbf{rep}(G^{(i)}) \leq \mathbf{rep}(G^{(i)}_{del}) + 2(i+1) \leq (\sqrt{4.5} + 2\cdot\sqrt{2\cdot\mathbf{eg}(\Sigma^{(i)})})\sqrt{n} + 2(i+1) \leq (\sqrt{4.5} + 2\cdot\sqrt{2\cdot\mathbf{eg}(\Sigma) - i})\sqrt{n} + 2(i+1) \leq (\sqrt{4.5} + 2\cdot\sqrt{2\cdot\mathbf{eg}(\Sigma)})\sqrt{n}$ for $n >> i$. □

We use Lemma 3.4.3 to extend the process of cutting along noncontractible tight nooses such that we obtain a planar graph with a small number of disjoint cut-nooses of small lengths. Let $g \leq \mathbf{eg}(\Sigma)$ be the number of iterations needed to cut along shortest noncontractible nooses such that they turn a Σ-embedded graph G into a planar graph G'. However, these cut-nooses may not be disjoint. In our dynamic programming approach we need pairwise disjoint cut-nooses. Thus, whenever we cut along a noose, we manipulate the cut-nooses found so far. After g iterations, we obtain the *set of cut-nooses* $\mathfrak{N}$ that is a set of disjoint cut-nooses bounding empty open disks in the embedding of G'. Let $L(\mathfrak{N})$ be the *length* of $\mathfrak{N}$ as the sum over the lengths of all cut-nooses in $\mathfrak{N}$.

Proposition 3.4.5. *It is possible to find, in polynomial time, a set of cut-nooses $\mathfrak{N}$ that contains at most $2g$ disjoint cut-nooses. Furthermore $L(\mathfrak{N})$ is at most $2g\,(\sqrt{4.5} + 2\cdot\sqrt{2\cdot\mathbf{eg}(\Sigma)})\sqrt{n}$.*

Proof. Let $\mathfrak{N}_i$ be the set of disjoint cut-nooses after i cuts. Consider the cases of Lemma 3.4.3 of how a shortest noncontractible (tight) noose N_s intersects a cut-noose of $\mathfrak{N}_i$.

• Suppose N_s intersects with the empty disk Δ_j bounded by $N_j \in \mathfrak{N}_i$. Let $P_1 \cup P_2 = N_j$ be the two partial nooses of N_j determined by the intersection of N_j and N_s. When we cut along N_s, we replace N_s by the contractible cut-nooses N_X and N_Y. We replace $N_X \cap \Delta_j$ by P_1 and $N_Y \cap \Delta_j$ by P_2. In $\mathfrak{N}_i$ we substitute N_j by N_X and N_Y. Note that N_s can intersect with several disjoint cut-nooses of $\mathfrak{N}_i$ in this way. See upper diagrams Figure 3.6 for an example.

• Suppose N_s intersects with $N_j \in \mathfrak{N}_i$ in one vertex. One of the cut-nooses N_X, N_Y intersects with N_j in vertex v. Delete v from N_j and add N_X, N_Y. to $\mathfrak{N}_i$. Also here N_s can intersect with several disjoint cut-nooses of $\mathfrak{N}_i$ in this way. See the middle diagrams in Figure 3.6 for an example.

• Suppose N_s intersects with $N_j \in \mathfrak{N}_i$ in two vertices x, y and $N_s \cap \Delta_j = \varnothing$ (corresponding to special case 2.2) in Lemma 3.4.3). Since there is no vertex in the part of N_s between x and y we are allowed to shift that part entirely inside of Δ_j. See the lower diagrams

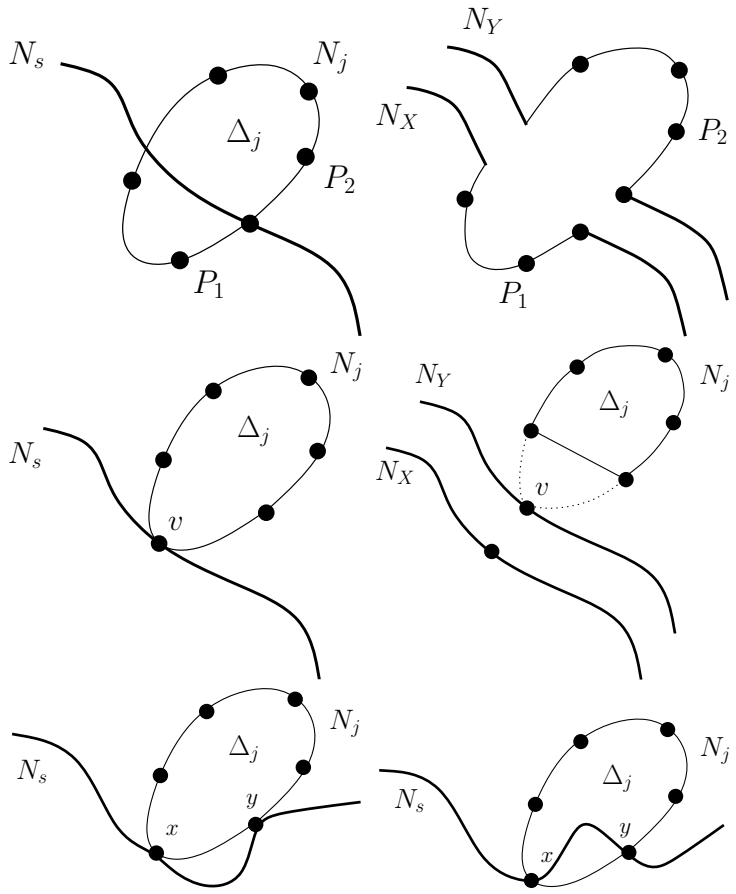


Figure 3.6: **Making cut-nooses disjoint.** The upper diagrams show how a noncontractible tight noose N_s partitions N_j into two partial nooses P_1 and P_2. $N_X \cup P_1$ and $N_Y \cup P_2$ form new cut-nooses. The middle diagrams show how N_s touches N_j in only one vertex v. Since N_j and N_Y intersect in v, we set $N_j == N_j \setminus v$. The lower diagrams show how to shift the part of N_s between vertices x and y from the outside of Δ_j into the inside.

in Figure 3.6 for an example. Thus, we obtain the first case above that N_s intersects with the empty disk Δ_j bounded by $N_j \in \mathfrak{N}_i$.

• Due to Lemma 3.4.4, the shortest noncontractible noose of any graph $G^{(i)}$ is bounded by $\sqrt{4.5} + 2 \cdot \sqrt{2 \cdot \mathbf{eg}(\Sigma)})\sqrt{n}$ where n is the number of vertices in the original graph G. Hence, $L(\mathfrak{N}) \leq 2g\,(\sqrt{4.5} + 2 \cdot \sqrt{2 \cdot \mathbf{eg}(\Sigma)})\sqrt{n}$.

□

We extend the definition of relaxed Hamiltonian sets from graphs embedded on a torus to graphs embedded on higher genus, i.e. from two cut-nooses N_X and N_Y to the set of cut-nooses $\mathfrak{N}$. For each vertex v in the vertex set $V(G)$ of graph G we define the vertex set D_v that contains all duplicated vertices $v_1, \ldots, v_f$ of v in $\mathfrak{N}$ along with v. Set $\mathfrak{D} = \bigcup_{v \in V(G)} D_v$.

Definition 3.4.6. *A set of disjoint paths* $\mathbf{P}$ *in* G' *is* relaxed Hamiltonian *if:*

(P1) Every path has its endpoints in $\mathfrak{N}$*.*

(P2) If a vertex $v_i \in D_v \in \mathfrak{D}$ *is an endpoint of path* P*, then there is one* $v_j \in D_v$ *that is also an endpoint of a path* $P' \neq P$*. All* $v_h \in D_v \setminus \{v_i, v_j\}$ *do not belong to any path.*

(P3) $v_i \in D_v$ *is an inner path vertex if and only if all* $v_h \in D_v \setminus \{v_i\}$ *are not in any path.*

(P4) Every vertex of the residual part of G' *is in some path.*

Again we define $\mathbf{HS}(G')$ to be the set of all relaxed Hamiltonian sets of G' and $\mathbf{HS}(G')/\sim$ to be the quotient set. Similar to torus-embedded graphs, we order the vertices of $\mathfrak{N}$ in a counterclockwise order $\pi_{\mathbf{L}}$ depending on the fixed paths between the cut-nooses of $\mathfrak{N}$:

Lemma 3.4.7. *Let* G' *be the planar graph after cutting along* $g \leq \mathbf{eg}(\Sigma)$ *tight nooses of* G *along with its set of disjoint cut-nooses* $\mathfrak{N}$*.* $|\mathbf{HS}(G')/\sim| = 2^{O(g\cdot(\log g + \sqrt{\mathbf{eg}(\Sigma)}\sqrt{n}))}$.

Proof. We create one cut-noose out of all the cut-nooses of $\mathfrak{N}$ by using "tunnels" as in the proof of Lemma 3.3.3 in the previous section. The difficulty here is that the cut-nooses are connected by a relaxed Hamiltonian set in an arbitrary way. By Lemma 3.1.4 in Section 3.1, we have with $|\mathfrak{N}| = O(g)$ and $L(\mathfrak{N}) = O(g\sqrt{\mathbf{eg}(\Sigma)}\sqrt{n})$ that $\texttt{q-paths}_{G'}(E(G'), V(G') \cap \mathfrak{N}) = 2^{O(g \log g)} \cdot 2^{O(g \log[g\sqrt{\mathbf{eg}(\Sigma)}\sqrt{n}])} \cdot 2^{O(g\sqrt{\mathbf{eg}(\Sigma)}\sqrt{n})} = 2^{O(g\cdot(\log g + \sqrt{\mathbf{eg}(\Sigma)}\sqrt{n}))}$. Since as in Lemma 3.3.3, $|\mathbf{HS}(G')/\sim|$ only differs by the additional case of property $(P3)$ of Definition 3.4.6, we have cardinality $|\mathbf{HS}(G')/\sim| = O^*(\texttt{q-paths}_{G'}(E(G'), V(G') \cap \mathfrak{N}))$. □

We use the tree structure, we obtain from the proof of Lemma 3.1.4, in order to cut the sphere along that structure. Given such a tree structure, we create tunnels in order to connect open disks and to merge them to one disk. Let $\mathbf{C}$ be a candidate of $\mathbf{HS}(G')/\sim$. Define graph H such that each cut-noose $N_i \in \mathfrak{N}$ in G' corresponds to a

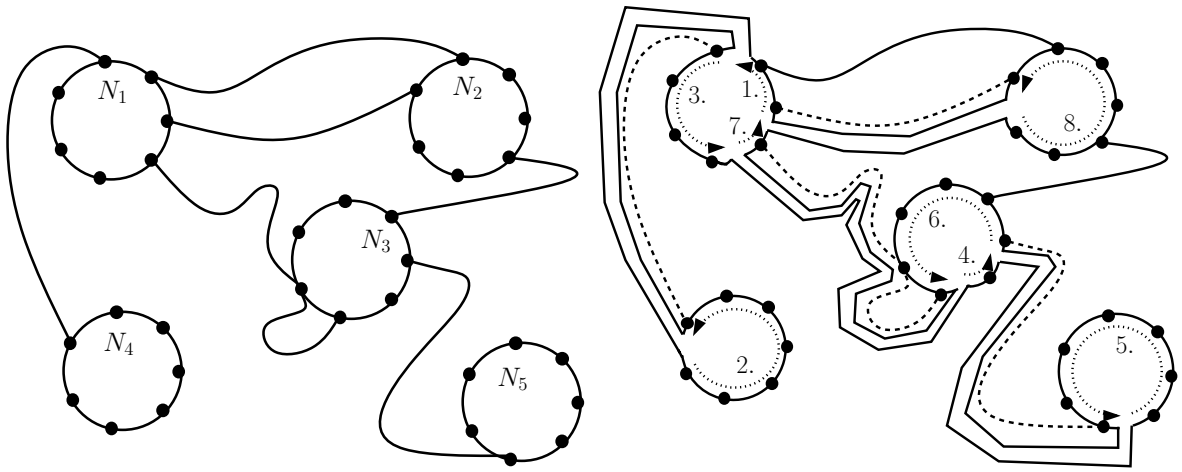


Figure 3.7: **Tree structure for fixing paths.** The left diagram shows a candidate connecting five cut-nooses $N_1, \ldots, N_5$ by paths. In the right diagram, the fixed paths are emphasized dashed. The nooses are connected by tunnels along these fixed paths. The order $\pi_{\mathbf{L}}$ of the vertices is illustrated by the labeled dotted and directed lines.

vertex i in $V(H)$. Two vertices i, j of H are adjacent if there is a path between vertices of N_i and N_j in $\mathbf{C}$. Let F be a spanning forest of H. For every pair of adjacent vertices i, j in F fix a path in G' between two arbitrary vertices $v_x^i \in N_i$ and $v_y^j \in N_j$. Walk along a tree by starting and ending in a node r and visiting all nodes by always visiting the next adjacent neighbor in counterclockwise order. A node is visited as many times as many neighbors it has. In this way we create tunnels in G' by ordering the vertices of the cut-nooses: Starting with an ordered list $\mathbf{L} = \{\varnothing\}$ and one cut-noose N_i and one endpoint $v_x^i \in N_i$ of a fixed path. Take in counterclockwise order the vertices of N_i into $\mathbf{L}$ that are between v_x^i and the last vertex before the next endpoint $v_y^i \in N_i$ connected to $v_z^j \in N_j$ in the fixed path $P_{i,j}$. Concatenate to $\mathbf{L}$ in counterclockwise order the vertices of N_j after v_z^j until the last vertex before the next endpoint of a fixed path. Repeat the concatenation until one reaches again v_x^i. Whenever an endpoint is visited for the second time concatenate it to $\mathbf{L}$, too. Create an ordered list $\mathbf{L}_C$ for every component C in $\mathbf{C}$ and concatenate $\mathbf{L}_C$ to $\mathbf{L}$. The order of $\mathbf{L}$ is then $\pi_{\mathbf{L}}$. See Figure 3.7 for an example.

Given the order $\pi_{\mathbf{L}}$ of the vertices $\mathfrak{N}$ for a candidate $\mathbf{C}$. As in the previous subsection, we preprocess the graph G' and fix the pairs of vertices of $\mathfrak{N}$ that form a path for candidate $\mathbf{C}$.

Lemma 3.4.8. *Let G' be the planar graph after cutting along $g \leq \mathbf{eg}(\Sigma)$ shortest non-contractible nooses of G. For a given sc-decomposition (T, μ, π) of G' of width ℓ and a candidate $\mathbf{C}$ the* Relaxed Hamiltonian Set *problem on G' can be solved in time $2^{O(g^2 \log \ell)} \cdot 2^{O(\ell)} \cdot n^{O(1)}$.*

The dynamic programming approach of the previous section can be extended without complicated encoding, which has the only purpose of giving better constants. We

give the detailed algorithm in Section 3.6, analyzing how the tight noose O_e can intersect several cut-nooses and showing that $V(O_e)$ still has Catalan structure.

Runtime analysis

Lemma 3.4.9. *Let G be a Σ-embedded graph with n vertices and G' the planar graph obtained after cutting along $g \leq \mathbf{eg}(\Sigma)$ tight nooses. Then, $\mathbf{bw}(G') \leq \sqrt{4.5} \cdot \sqrt{n} + 2g$.*

Proof. We only give an idea of the proof, that is extending the proof of Lemma 3.3.5. Here we delete temporarily all cut-nooses of $\mathfrak{N}$ and construct a sc-decompositionof G' of width at most $\sqrt{4.5} \cdot \sqrt{n}$. Now we simply make use of the argument that a middle set O_e intersects a cut-noose in at most two vertices for each cut-noose separately. Thus, we obtain at most two vertices more for O_e per cut-noose that O_e intersects. □

Lemmata 1.2.4, 3.4.7, 3.4.8 and 3.4.9 imply the following:

Theorem 3.4.10. *Given a Σ-embedded graph G on n vertices and $g \leq \mathbf{eg}(\Sigma)$. The* Hamiltonian cycle *problem on G can be solved in time $n^{O(g^2)} \cdot 2^{O(g\sqrt{g \cdot n})}$.*

3.5 Graphs excluding a fixed-sized minor

We know extend our algorithm to a class of graphs that is including the previous ones, the H-minor-free graphs, and show how to extend our notion of Catalan Structure to it. The core of our algorithm is a structure theorem [85] capturing the structure of all graphs excluding a fixed minor, also called H-minor-free graphs. That is, the theorem says that every such graph can be decomposed into a collection of graphs, each of which can "almost" be embedded into a surface of bounded genus, and combined in a tree-like structure. We show at the example of k-Longest Path how to obtain a parametrized algorithm of subexponential runtime. Note, that this algorithm can be reduced to the problem on torus-embedded and other bounded-genus graphs.

3.5.1 A cornerstone theorem of Graph Minors.

Given two graphs G_1 and G_2 and two h-cliques $S_i \subseteq V(G_i)$, $(i = 1, 2)$. We obtain graph G by identifying S_1 and S_2 and deleting none, some or all clique-edges. Then, G is called the *h-clique-sum* of the *clique!h-clique-sum!clique-sum components* G_1 and G_2. Note that the clique-sum gives many graphs as output depending on the edges of the clique that are deleted. According to Lemma 3.6.20 , given a graph G with branch-decomposition (T, τ), for any clique with vertex set S there exists a node $t \in T$ such that $S = \mathrm{mid}(\{t, a\}) \cup \mathrm{mid}(\{t, b\}) \cup \mathrm{mid}(\{t, c\})$ where a, b, c are the neighbors of t in T. We call such a vertex of T a *clique node* of S.

Let Σ be a surface. We denote as Σ^{-r} the subspace of Σ obtained if we remove from Σ the interiors of r disjoint closed disks (we will call them *vortex disks*). Clearly, the boundary $\mathbf{bor}(\Sigma^{-r})$ of Σ^{-r} is the union of r disjoint cycles. We say that G is *h-almost embeddable* in Σ if there exists a set $A \subseteq V(G)$ of vertices, called *apices of* G, where $|A| \leq h$ and such that $G - A$ is isomorphic to $G_u \cup G_1 \cup \cdots \cup G_r, r \leq h$ in a way that the following conditions are satisfied (the definition below is not the original one from [85] but equivalent, slightly adapted for the purposes of our paper):

• There exists an embedding $\sigma : G_u \to \Sigma^{-r}, r \leq h$ such that only vertices of G_u are mapped to points of the boundary of Σ^{-r}, i.e. $\sigma(G_u) \cap \mathbf{bor}(\Sigma^{-r}) \subseteq V(G)$ (we call G_u the *underlying graph* of G).
• The graphs $G_1, \ldots, G_r$ are pairwise disjoint (called *vortices* of G). Moreover, for $i = 1, \ldots, r$, if $E_i = E(G_i) \cap E(G_u)$ and $B_i = V(G_i) \cap V(G_u)$ (we call B_i *base set* of the vortex G_i and its vertices are the *base vertices* of G_i), then $E_i = \varnothing$ and $\sigma(B_i) \subseteq C_i$ where $C_1, \ldots, C_r$ are the cycles of $\mathbf{bor}(\Sigma^{-r})$.
• for every $i = 1, \ldots r$, there is a trunk decomposition $\mathcal{X}_i = (X_1^i, \ldots, X_{q_i}^i)$ of the vortex G_i with width at most h and a subset $J_i = \{j_1^i, \ldots, j_{|B_i|}^i\} \subseteq \{1, \ldots, q_i\}$ such that $\forall_{k=1,\ldots,|B_i|}\ u_k^i \in X_{j_k^i}^i$ for some respectful ordering $(u_1^i, \ldots, u_{|B_i|}^i)$ of B_i. (An ordering $(u_1^i, \ldots, u_{|B_i|}^i)$ is called *respectful* if the ordering $(\sigma(u_1^i), \ldots, \sigma(u_{|B_i|}^i))$ follows the cyclic ordering of the corresponding cycle of $\mathbf{bor}(\Sigma^{-r})$.) For every vertex $u_k^i \in B_i$, we call $X_{j_k^i}^i$ *overlying set* of u_j^i and we denote it by $\mathbf{X}(u_j^i)$.

If in the above definition $A = \varnothing$, then we say that G is *smoothly h-almost embeddable* in Σ. Moreover, if $r = 0$, then we just say that G is *embeddable* in Σ.

For reasons of uniformity, we will extend the notion of the overlying set of a vertex in B_i to any other vertex v of the underlying graph G_u by defining its *overlying set* as the set consisting only of v. For any $U \subseteq V(G_u)$, the *overlying set* of U is defined by the union of the overlying sets of all vertices in U and it is denoted as $\mathbf{X}(U)$.

We will strongly use the following structural theorem of Robertson and Seymour (see [85],) characterizing H-minor-free graphs.

Proposition 3.5.1 ([85])**.** *Let $\mathcal{G}$ be the graph class not containing a graph H as a minor. Then there exists a constant h, depending only on H, such that any graph $G \in \mathcal{G}$ is the (repeated) h-clique-sum of h-almost embeddable graphs (we call them* clique-sum components*) in a surface Σ of genus at most h.*

That is, beginning with an h-almost embeddable graph G, we repeatedly construct the h-clique-sum of G with another h-almost embeddable graph.

3.5.2 k-Longest Path

Before we state the main result of this section, we need some notation especially for the context of our algorithm. Given a graph H and a function f we use the notation

$O_H(f)$ to denote $O(f)$ while emphasizing that the hidden constants in the big-O notation depend exclusively on the size of H. We also define analogously the notation $\Omega_H(f)$.

We change the definition of branch-decomposition having the Catalan structure of Section 3.1.2 slightly for H-minor-free graphs. Given a graph G and a branch-decomposition (T, τ) of G, we say that (T, τ) has the *Catalan structure* if

$$\text{for any edge } e \in E(T),\ \texttt{q-paths}(E_e, \partial E_e) \leq 2^{O_H(|\partial E_e|)}.$$

Our main result is the following.

Theorem 3.5.2. *For any H-minor-free graph class $\mathcal{G}$, the following holds:*
For every graph $G \in \mathcal{G}$ and any positive integer w, it is possible to construct a time $O_H(1) \cdot n^{O(1)}$ algorithm that outputs one of the following:

1. *A correct report that G contains a $(w \times w)$-grid as a minor.*
2. *A branch-decomposition (T, τ) with the Catalan structure and of width $O_H(w)$.*

In what follows, we will give the description of the algorithm of Theorem 3.5.2 and we will sketch the main arguments supporting its correctness. The full proof is lengthy and complicated and all lemmata supporting its correctness can be found in Section 3.6.

1. Use the time $O_H(1) \cdot n^{O(1)}$ algorithm of [27] to decompose the input graph into a collection $\mathcal{C}$ of clique-sum components as in Proposition 3.5.1. Every graph in $\mathcal{C}$ is a $\mathbf{eg}_H$-almost embeddable graph to some surface of genus $\leq \mathbf{eg}_H$ where $\mathbf{eg}_H = O_H(1)$.
2. For every $G^a \in \mathcal{C}$, do
  a. Let G^s be the graph G^a without the apex vertices A (i.e. G^s is smoothly $\mathbf{eg}_H$-almost embeddable in a surface of genus $\mathbf{eg}_H$). Denote by G^s_u the underlying graph of G^s.
  b. Set $G^{(1)}_u \leftarrow G^s_u$, $G^{(1)} \leftarrow G^s$, and $i \leftarrow 1$.
  c. Apply the following steps as long as $G^{(i)}_u$ is non-planar.
    i. Find a non-contractible noose N in $G^{(i)}_u$ of minimum length, using the polynomial time algorithm in [93].
    ii. If $|N| \geq 2^{i-1} f(H) \cdot w$ then output "*The input graph contains a $(w \times w)$-grid as a minor*" and stop. The estimation of $f(H)$ comes from the results in [34], presented in Lemma 3.6.9. Along with Lemma 3.6.10 follows the correctness of this step.
    Notice that, by minimality, N cannot intersect the interior of a hole or a vortex disk Δ more than once and, for the same reason, it can intersect $\mathbf{bor}(\Delta)$ in at most two vertices. If $\mathbf{int}(\Delta) \cap N = \varnothing$ and $\mathbf{bor}(\Delta) \cap N = \{v, w\}$, again from minimality, v and w should be successive in $\mathbf{bor}(\Delta)$. In this case, we re-route this portion of N so that it crosses the interior of Δ (see Figure 3.8).
    iii. As long as N intersects some hole (initially the graph $G^{(1)}$ does not contain holes but they will appear later in $G^{(i)}$'s for $i \geq 2$) or some vortex disk of $G^{(i)}_u$ in *only* one vertex v, update $G^{(i)}$ by removing v and the overlying set of all its relatives (including $\mathbf{X}(v)$) from $G^{(1)}, \ldots, G^{(i)}$. To maintain the $O_H(1)$-almost embeddability of G^a, compensate this loss of vertices in the initial graph $G^s = G^{(1)}$, by moving in

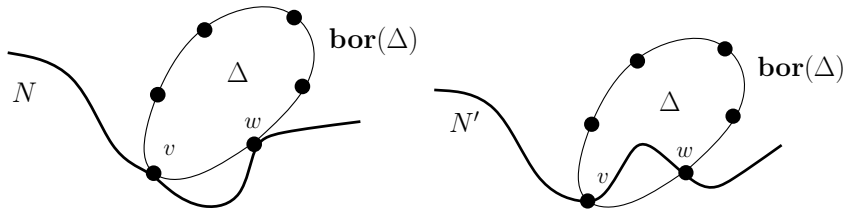


Figure 3.8: Re-routing a noose.

A the overlying set of the relative of v in $G^{(1)}$ (as the number of vortex disks and holes depends only on H, the updated apex set has again size depending on H). Notice that after this update, all cut-nooses found so far, either remain intact or they become smaller. The disks and vortices in $G^{(1)}, \ldots, G^{(i)}$ may also be updated as before and can only become smaller. We observe that after this step, if a noose N intersects a hole or vortex disk Δ it also intersects its interior and therefore it will split Δ into two parts Δ_1 and Δ_2.

iv. Cut $G_u^{(i)}$ along N and call the two disks created by the corresponding cut of the surface *holes* of the new embedding. We go through the same cut in order to "saw" $G^{(i)}$ along N as follows: If the base set of a vortex is crossed by N then we also split the vortex according to the two sides of the noose; this creates two vortices in $G^{(i+1)}$. For this, consider a vortex G^v and a trunk decomposition $\mathcal{X} = (X_1, \ldots, X_q)$ of G^v. Let also a, b be the vertices of the base set B of G^v that are intersected by N and let $a \in X_{j_a}$, $b \in X_{j_b}$, where w.l.o.g. we assume that $a < b$. When we split, the one vortex is the subgraph of G^v induced by $X_1 \cup \cdots \cup X_{j_a} \cup X_{j_b} \cup \cdots \cup X_q$ the other is the subgraph of G^v induced by $X_{j_a} \cup \cdots \cup X_{j_b}$ (notice that the vertices that are duplicated are those in X_{j_a} and X_{j_b}). Let $G^{(i+1)}$ be the graph embedding that is created that way and let $G_u^{(i+1)}$ be its underlying graph . Recall that, from the previous steps, a vortex disk or a hole Δ (if divided) is divided into two parts Δ_1 and Δ_2 by N. That way, the splitting of a vortex in $G^{(i)}$ creates two vortices in $G^{(i+1)}$. As the number of vortices in $G^{(i)}$ is $O_H(1)$, the same holds also for the number of vortices in $G^{(i+1)}$. If N splits a hole of $G^{(i+1)}$, then the two new holes Δ'_1, Δ'_2, that the splitting creates in $G^{(i+1)}$, are augmented by the two parts Δ_1 and Δ_2 of the old hole Δ (i.e. $\Delta'_j \leftarrow \Delta_j \cup \Delta'_j, j = 1, 2$).

v. $i \leftarrow i + 1$.

The loop of step **2.c.** ends up with a planar graph $G_u^{(i)}$ after $O_H(1)$ splittings because the genus of $G_u^{(1)}$ is $O_H(1)$ (each step creates a graph of lower Euler genus – see [74, Propositions 4.2.1 and 4.2.4]). This implies that the number of holes or vortex disks in each $G_u^{(i)}$ remains $O_H(1)$. Therefore, $G^{(i)}$ is a smoothly $O_H(1)$-almost embeddable graph to the sphere. Also notice that the total length of the holes of $G_u^{(i)}$ is upper bounded by the sum of the lengths of the nooses we cut along, which is $O_H(w)$.

d. Set $G^p \leftarrow G^{(i)}$ and $G_u^p \leftarrow G_u^{(i)}$ and compute an optimal sphere-cut decomposition (T_u^p, τ_u^p) of G_u^p, using the polynomial algorithm from [89].

e. If $\mathbf{bw}(G_u^p) \geq 2^{\mathbf{eg}_H} \cdot f(H) \cdot w = \Omega_H(w)$, then output "*The input graph contains a*

($w \times w$)-*grid as a minor*" and stop. This step is justified by Lemma 3.6.11.

f. Enhance (T^p_u, τ^p_u), so that the edges of the vortices of G^p are included to it, as follows: Let G^v be a vortex of G^p with base set $B = \{u_1, \ldots, u_m\}$ ordered in a respectful way such that $\forall_{k=1,\ldots,m} u_k \in \text{mid}(f_{j_k})$ where the ordering $f_{j_1}, \ldots, f_{j_m}$ contains the edges of a longest path of the tree T^* of some trunk decomposition (T^*, τ^*) of G^v. Update (T^p_u, τ^p_u) to a branch-decomposition $(\hat{T}^p_u, \hat{\tau}^p_u)$ of $\hat{G}^p$ (if the branchwidth of a graph is more than 1, it does not change when we add loops). Let $l_1, \ldots, l_m$ be the leaves of $(\hat{T}^p_u, \hat{\tau}^p_u)$ corresponding to the loops of the base vertices of G^v. We subdivide each f_{j_k} in T^* and we identify the subdivision vertex with l_k for any $k = 1, \ldots, m$. We make the resulting graph a ternary tree, by removing a minimum number of edges in T^* and devolving their endpoints in the resulting forest. That way, we construct a branch-decomposition of $\hat{G}^p_u \cup G^v$ which, after discarding the leaves mapped to loops, gives a branch-decomposition of $G^p_u \cup G^v$ (see Figure 3.9). Applying this transformation for

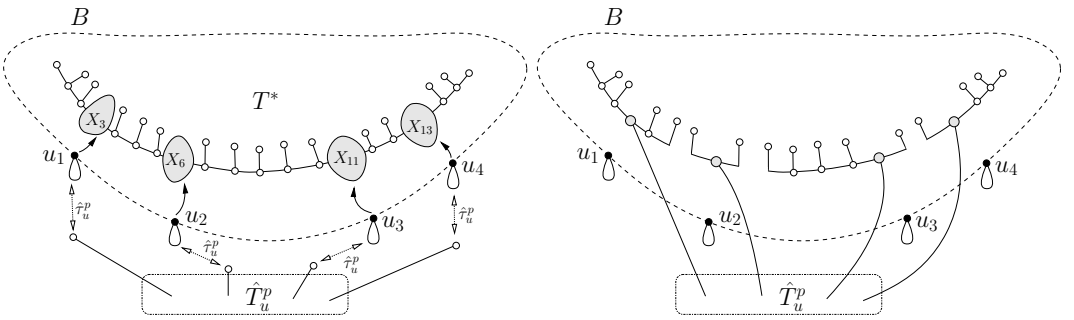


Figure 3.9: The procedure of enhancing the branch-decomposition (T^p_u, τ^p_u) of G^p_u to a branch-decomposition of G^p.

each vortex G^v of G^p, we construct a branch-decomposition of G^p. In Lemma 3.6.12, we prove that this enhancement of the branch-decomposition of G^p_u can add $O_H(1)$ vertices for each vertex in $\text{mid}(e), e \in E(T^p_u)$, therefore, $\mathbf{bw}(G^p) = O_H(\mathbf{bw}(G^p_u))$.

g. Notice that, while successively splitting G^s during the loop of step **2.c.**, all edges remain topologically intact (only vertices may be duplicated). This establishes a bijection between $E(G^s)$ and $E(G^p)$, which allows us to transform (T^p, τ^p) to a branch-decomposition (T^s, τ^s) where $T^s = T^p$. At this point, we have to prove that if the bounds of Theorem 3.5.2 holds for the graph $G^p = G^{(i)}$ (a graph that is smoothly $O_H(1)$-almost embeddable in the sphere), then they also hold for the graph $G^s = G^{(1)}$ that is a smoothly $O_H(1)$-almost embeddable in a surface of higher genus. We prove that $\mathbf{bw}(G^s) = O_H(\mathbf{bw}(G^p))$ with the help of Lemma 3.6.13.a (using induction). However, what is far more complicated is to prove that (T^s, τ^s) has the Catalan structure. For this, we first prove (using inductively Lemma 3.6.13.b) that for any edge e of $T^s = T^p$, holds that

$$\texttt{q-paths}_{G^s}(E_e, \partial E_e) \;\leq\; \texttt{q-paths}_{G^p}(E_e, \partial E_e \cup D_e), \tag{3.13}$$

where D_e is the set of all vertices of the holes of G^p that are endpoints of edges in E_e. Intuitively, while splitting the graph G^s along non-contractible nooses, the split

vertices in the nooses (i.e., the vertices in D_e) may separate paths counted in the left side of Equation (3.13). Therefore, in order to count them, we have to count equivalence classes of collections of internally vertex disjoint paths in the planar case allowing their endpoints to be not only in ∂E_e but also in D_e. That way, we reduce the problem of proving that (T^s, τ^s) has the Catalan structure, to the following problem: find a bound for the number of equivalent classes of collections of vertex disjoint paths whose endpoints may be a) vertices of the disk Δ_e bounding the edges E_e in the sphere-cut decomposition (T^p, τ^p) of G^p_u, along with their overlying sets (all-together, these ends are at most $\mathbf{bw}(G^p) = O_H(w)$, because of Lemma 3.6.12) and b) vertices (and their overlying sets) of $2 \cdot \mathbf{eg}_H = O_H(1)$ disjoint holes (created by cutting along nooses) of total size $\leq \mathbf{eg}_H \cdot (2^{\mathbf{eg}_H} - 1) \cdot f(H) \cdot w = O_H(w)$ (see the proof of Lemma 3.6.10). Notice that these paths can be routed also via $\leq \mathbf{eg}_H \cdot 2^i \leq \mathbf{eg}_H \cdot 2^{\mathbf{eg}_H} = O_H(1)$ vortices (because, initially, G^s_u had $\mathbf{eg}_H$ vortices and, in the worst case, each noose can split every vortex into two parts), each of unbounded size. Recall that the holes and the vortex disks of G^p_u do not touch (i.e. they intersect but their interiors do not) because of the simplification in Step **2.c.iii**, however, they may have common interiors. Finally, the boundary of Δ_e can touch any number of times a vortex disk or a hole but can traverse it only once (recall that by the definition of sc-decompositions $\mathbf{bor}(\Delta_e)$ should be a tight noose). (For an example of the situation of the holes and vortices around the disk bounding the edges E_e, see Figure 3.14.) Our target is to relate $\texttt{q-paths}_{G^p}(E_e, \partial E_e \cup D_e)$ to the classical Catalan structure of non-crossing partitions on a cycle. As this proof is quite technical, it is stated subsequent to this section (Lemma 3.6.14) and, here, we will give just a sketch. Our first two steps are to "force" holes and vortex disks not to touch the boundary of Δ_e and to "force" vortex disks not to intersect with holes or with $\mathbf{bor}(\Delta_e)$. For each of these two steps, we bound $\texttt{q-paths}_{G^p}(E_e, \partial E_e \cup D_e)$ by its counterpart in a "normalized" instance of the same counting problem (related to the original one by a "rooted minor" relation). That way, the problem is reduced to counting equivalent classes of collections of vertex disjoint paths with endpoints (recall that there are $O_H(w)$ such endpoints) on the boundary of $O_H(1)$ disjoint holes (the disk taken if we remove from Δ_e all holes that intersect it, is also considered as one of these holes). However, we still do not have to count equivalent classes of non-crossing collections of paths because of the presence of the vortices that may permit crossing paths. At that point, we prove that no more than $O_H(\beta)$ paths can mutually cross, where $\beta = O_H(1)$ is the number of vortices. Using this observation, we prove that each equivalent class is the superposition of $O_H(1)$ equivalent classes of non-crossing collections of disjoint paths. Because of this, the number of equivalent classes of collections of disjoint paths are in total $2^{O_H(w)}$ and, that way, we bound $\texttt{q-paths}_{G^p}(E_e, \partial E_e \cup D_e)$ as required.

h. Construct a branch-decomposition (T^a, τ^a) of G^a by adding in (T^s, τ^s) the edges incident to the apices of G^a. To do this, for every apex vertex a and for every neighbor v, choose an arbitrary edge e of T^s, such that $v \in \partial E_e$. Subdivide e and add a new edge to the new node and set $\tau(\{a, v\})$ to be the new leaf. The proof that $\mathbf{bw}(G^a) = O_H(\mathbf{bw}(G^s))$ is easy, as G^s contains only $O_H(1)$ vertices more (Lemma

3.6.14). With more effort (and for the same reason) we prove that the Catalan structure for G^s implies the Catalan structure for G^a (Lemma 3.6.19).

3. For any $G^a \in \mathcal{C}$, merge the branch-decompositions constructed above according to the way they are joined by clique sums and output the resulting branch-decomposition of the input graph G. In particular, if (T_1^a, τ_1^a) and (T_2^a, τ_2^a) are two branch-decompositions of two graphs G_1^a and G_2^a with cliques S_1 and S_2 respectively and $|S_1| = |S_2|$, we construct a branch-decomposition (T', τ') of the graph G', taken after a clique sum of G_1^a and G_2^a, as follows: Let t^i be a clique-node of S_i in $(T_i, \tau_i), i = 1, 2$. Then, the branch-decomposition (T', τ') of G' is obtained by first subdividing an incident edge $e_{t^i}, i = 1, 2$ and then connecting the new nodes together. Secondly, remove each leaf l of T' that corresponds to an edge that has a parallel edge or is deleted in the clique-sum operation and finally contract an incident edge in T' of each degree-two node. We prove (Lemma 3.6.23) that this merging does not harm neither the bounds for branchwidth nor the Catalan structure of the obtained branch-decomposition and this finally holds for the input graph G, justifying Theorem 3.5.2.

3.5.3 Algorithmic consequences

A first application of Theorem 3.5.2 is the following.

Corollary 3.5.3. *The problem of checking whether there is a path of length k on H-minor-free graphs can be solved in $2^{O_H(\sqrt{k})} \cdot n^{O(1)}$ steps.*

Proof. We apply the algorithm of Theorem 3.5.2 for $w = \sqrt{k}$. If it reports that G contains a $(\sqrt{k} \times \sqrt{k})$-grid, then G also contains a path of length k. If not, then the algorithm outputs a branch-decomposition (T, τ) of width $O_H(\sqrt{k})$, as in Theorem 3.5.2. By applying dynamic programming on (T, τ) we have, for each $e \in E(T)$, to keep track of all the ways the required path (or cycle) can cross $\text{mid}(e) = \partial E_e$. This is proportional to $\texttt{q-paths}_G(E_e, \partial E_e)$ (counting all ways these paths can be rooted through ∂E_e). As $\texttt{q-paths}_G(E_e, \partial E_e) = 2^{O_H(\sqrt{k})}$ we have the claimed bounds. □

Note, that for $k = \log^2 n$, Corollary 3.5.3 gives a polynomial time algorithm for checking if a n-vertex graph has a path of length $\log^2 n$.

Other problems that can be solved in $2^{O_H(\sqrt{k})} \cdot n^{O(1)}$ steps in H-minor-free graph classes, applying simple modifications to our technique, are the standard parameterizations of Longest Cycle, Feedback Vertex Set, and Cycle/Path Cover (parameterized either by the total length of the cycles/paths or the number of the cycles/paths).

Moreover, combining Theorem 3.5.2, with the results in [38] we can derive time $2^{O_H(\sqrt{k})} \cdot n^{O(1)}$ algorithms for problems emerging from contraction closed parameters for apex-minor-free graph classes (a graph is an *apex* graph the removal of one of its vertices creates a planar graph). The most prominent examples of such problems are Connected Dominating Set and Max Leaf Tree. (The best previous algorithm for

these problems for apex-minor-free graph classes was a $2^{O_H(\sqrt{k}\cdot\log k)}\cdot n^{O(1)}$ step algorithm given in [35].)

Our technique can also be used to design fast subexponential exact algorithms. Notice that the branchwidth of any H-minor-free graph is at most $O_H(\sqrt{n})$ [9]. The algorithm of Theorem 3.5.2 will output a branch-decomposition of width $O_H(\sqrt{n})$ that, using an adequate definition of Catalan structure, can be used to derive $2^{O_H(\sqrt{n})}$ step algorithms for several problems. Consider for example Weighted Graph Metric TSP (TSP with the shortest path metric of G as distance metric). We saw in Section 3.2.3 how to solve Graph Metric TSP on planar graphs.The basic idea is that any solution to Graph Metric TSP can be reduced to finding a minimum weight spanning Eulerian subgraph. In this case, instead of having collections of paths $\mathbf{paths}_G(E_e, \partial E_e)$ we deal with connected components, say $\mathbf{comp}_G(E_e, \partial E_e)$. Nevertheless, we can use the Catalan structures argument and extend our counting results about $\texttt{q-paths}_G(E_e, \partial E_e)$. Apart from the problems that we have already mentioned above, $2^{O_H(\sqrt{n})}$ step exact algorithms can be designed for Steiner Tree, Maximum Full Degree Spanning tree, and other types of spanning tree problems.

3.6 Appendix

3.6.1 Proof of Lemma 3.3.4

Preprocess G'. In a preprocessing step we delete all vertices of $N_X \cup N_Y$ from G' which do not belong to any path of the candidate $\mathbf{C} = (\mathbf{T}, \mathbf{I})$. The other vertices in $N_X \cup N_Y$ are encoded by *base values* $\{[,], S, \square\}$. This *base encoding* depends on the order π_{XY} and is fixed throughout the phase of dynamic programming. Assume that for the tuple (s, t) of $\mathbf{T}$, s is marking the first vertex in π_{XY} and t the last vertex. We encode both s and t by 'S'. For every other tuple (s_i, t_i) of $\mathbf{T}$ we encode s_i by '[' and t_i by ']' where $\pi_{XY}(s_i) < \pi_{XY}(t_i)$.The additional value '$S$' is important for a consistent dynamic programming. It determines the "tunnel" created by the path with endpoints s and t. As described in the proof of Lemma 3.3.3 the cut-nooses N_X, N_Y and the tunnel border one face that we treat as an outer face and so enables the base encoding. The vertices of $\mathbf{I}$ simply are encoded by base value '$\square$'.

Constructing branch-decomposition. We use Theorem 2.3.1 to compute in polynomial time a sc-decomposition (T, μ, π) of G' of optimum width ℓ. For dynamic programming we root T at an arbitrary node r. We proceed as for dynamic programming in Section 3.2. In the following, we often do not distinguish between $\mathbf{mid}(e)$ and $O_e \cap V(G)$. We start at the leaves of T and work 'bottom-up' processing the subgraphs rooted at the edges up to the root edge. Consider the intersection of relaxed Hamiltonian set $\mathbf{P}$ and subgraph G_e. All paths of $\mathbf{P} \cap G_e$ start and end in O_e and $G_e \cap (N_X \cup N_Y)$. For a dynamic programming step we need the information on how a tight noose O_e and

$N_X \cup N_Y$ intersect and which parts of $N_X \cup N_Y$ are a subset of the subgraph G_e. Define the set of O-arcs $\mathcal{X} = (G_e \setminus O_e) \cap (N_X \cup N_Y)$.

In the dynamic programming approach we differentiate three phases. In Phase 1 no vertex of $N_X \cup N_Y$ is contained in disk Δ_e bounded by O_e, thus $\mathcal{X} = \varnothing$. Note that in this phase $(N_X \cup N_Y) \cap O_e$ is not necessary empty because O_e may touch N_X,N_Y in common vertices. Hence all paths must start and end in O_e. At some step of dynamic programming we arrive at Phase 2: $\mathcal{X} \neq \varnothing$ but neither $V(N_X) \subseteq \mathcal{X}$, nor $V(N_Y) \subseteq \mathcal{X}$. This is when we connect the *loose* paths—i.e. paths with endpoints in $O_e \setminus (N_X \cup N_Y)$—to their predestinated endpoints in N_X and N_Y. Finally, we arrive at the situation, Phase 3, when either $V(N_X) \subseteq \mathcal{X} \vee V(N_Y) \subseteq \mathcal{X}$, or $V(N_X) \subseteq \mathcal{X} \wedge V(N_Y) \subseteq \mathcal{X}$. The situation that eventually occurs in Phase 3 is that a cut-noose is entirely part of G_e but not intersecting O_e. We have to draw special attention to the fact that the relative position of the cut-noose and O_e are not known. By working out some special cases, we take care that the residual paths with one determined endpoint in N_X and N_Y and the other endpoint $O_e \setminus (N_X \cup N_Y)$ in are connected in the right way.

Proposition 3.6.1. *Phase 1: Given an sc-decomposition (T, μ, π) of G' of width ℓ and a candidate $\mathbf{C} = (\mathbf{T}, \mathbf{I})$. The phase of dynamic programming with $\mathcal{X} = \varnothing$ takes time $O(2^{3.292\ell})$.*

Every vertex of the subgraph G_e below O_e is part of one of the vertex-disjoint paths $P_1, \ldots, P_q$ of $\mathbf{P} \cap G_e$ with endpoints in O_e. The state of dynamic programming is specified by an ordered ℓ-tuple $\vec{t_e} := (v_1, \ldots, v_\ell)$ with the variables $v_1, \ldots, v_\ell$ corresponding to the vertices of $O_e \cap V(G)$. The variable set $\mathbf{V}$ has four values:$\mathbf{V} = \{0, 1_[, 1_], 2\}$. For edge e, we define the *assignment* $c_e : V(O_e) \to \mathbf{V}$. For every state, we compute a Boolean value $B_e(v_1, \ldots, v_\ell)$ that is TRUE if and only if $P_1, \ldots, P_q$ have the following properties: *(A1) Every vertex of $V(G_e) \setminus O_e$ is contained in one of the paths P_i, $1 \leq i \leq q$.*

(A2) Every P_i has both its endpoints in $O_e \cap V(G)$;

We obtain exactly the same algorithm as in Section 3.2 for PLANAR HAMILTONIAN CYCLE.

We count the total cost of forming an ℓ-tuple from O_P by summing over i: the number of $1_[$'s and $1_]$'s in the assignments for I:

$$C(\ell) = \sum_{i=0}^{\frac{\ell}{2}} \binom{\frac{\ell}{2}}{i} 2^{\frac{\ell}{2}-i} Q(\ell, i)^2 \approx (4\sqrt{6})^\ell \approx 2^{3.292\ell} \tag{3.14}$$

There is one restriction to the encoding of the vertices in $O_e \cap (N_X \cup N_Y)$: a vertex with base value '[',']' or 'S' cannot be assigned with '2' at any stage.

Proposition 3.6.2. *Phase 2: The phase of dynamic programming with $\mathcal{X} \neq \varnothing$ and $V(N_X) \nsubseteq \mathcal{X}$ and $V(N_Y) \nsubseteq \mathcal{X}$ takes time $O(2^{6.360\ell})$.*

Let us remind that $\mathcal{X} = (G_e \setminus O_e) \cap (N_X \cup N_Y)$. The difficulty of the second phase lies in keeping track of the base encoding of the vertices in $\mathcal{X}$. Thus, we do not want to memorize explicitly with which endpoint in $\mathcal{X}$ a vertex of O_e forms a path. We apply again the Catalan structures. The key to it is firstly that the vertices of O_e inherit the base values of the sets $\mathbf{T}$ and $\mathbf{I}$ of candidate $\mathbf{C}$.That is, if one vertex of O_e is paired with a vertex assigned by '[' it must be paired in $\overline{G_e}$ with a vertex with value ']'. Secondly, we observe that the cut-nooses $N_X \cup N_Y$ and the tight noose O_e intersect in a characteristic way: we show that we obtain a structure that allows us to encode paths in an easy way. In other words we make use of the structure of the subgraph G_e bordered by $N_X \cup N_Y$ and O_e for synchronizing the encoding of $\mathbf{T}$ and $\mathbf{I}$ with the encoding of O_e. To do so, we need some more definitions. We define a *partial noose* as a proper connected subset of either the tight noose O_e or one of the cut-nooses N_X, N_Y. A partial noose is bounded by vertices or points of the intersection $O_e \cap (N_X \cup N_Y)$. A *partial component* of a graph is embedded on an open disk bounded by partial nooses. The vertices in the intersection of two partial components are called *connectors*. In fact, G_e can be partitioned into several partial components with no connector in three components. Each component is bordered by partial nooses of $N_X \cup N_Y$ and O_e.

Proposition 3.6.3. *The subgraph G_e is the union of partial components $C_1, \ldots, C_q$ $(q \geq 1)$ such that for every i*

$C_i \cap (\bigcup_{r=1, r\neq i}^{q} C_r) \subseteq O_e \cap (N_X \cup N_Y)$. *Furthermore, for every i, j, h, $C_i \cap C_j \cap C_h = \varnothing$.*

Proof. Recall that by definition a tight noose intersects a region exactly once. Hence O_e intersects at most once the empty disks Δ_X and Δ_Y that are bounded by N_X and N_Y. In this case, O_e and $(N_X \cup N_Y)$ can intersect in vertices or as well once or twice the arc between to successive vertices in N_X and N_Y, respectively. That is due to the fact that N_X and N_Y are cut-nooses and hence may have several arcs in one face. In contrast, O_e and $(N_X \cup N_Y)$ can touch arbitrarily often, but only in vertices. In Phase 2, $\Delta_e \cap (\Delta_X \cup \Delta_Y) \neq \varnothing$. Hence, if one removes $\Delta_X \cup \Delta_Y$ then Δ_e is partitioned into several disks $\Delta_1, \ldots, \Delta_q$ each bordered by the union of some partial nooses bounded by vertices v of $V(O_e) \cap V(N_X \cup N_Y)$ or some points μ of the crossing of the arcs between two successive vertices of O_e and $N_X \cup N_Y$. Since N_X and N_Y do not intersect, we have that v and μ, respectively, is an endpoint of at most four partial nooses. Hence, v is neighboring at most two partial components and μ one. □

See the left diagram of Figure 3.10 for an example.

Proposition 3.6.4. *Each C_i is bordered by sets of partial nooses A_i and B_i with $A_i \subset (N_X \cup N_Y) \setminus O_e$ and $B_i \subset O_e$ with $\bigcup_{i=1}^{k} A_i \cup B_i = G_e \cap ((N_X \cup N_Y) \cup O_e)$ such that one of the following hold:*

1. *$|A_i| = |B_i| = 1$ with $A_i \subset N_X$ or $A_i \subset N_Y$,*
2. *$|A_i| = |B_i| = 2$ with $A_i \subset N_X$ and $A_i \subset N_Y$.*

There is at most one partial component C_i with property 2.

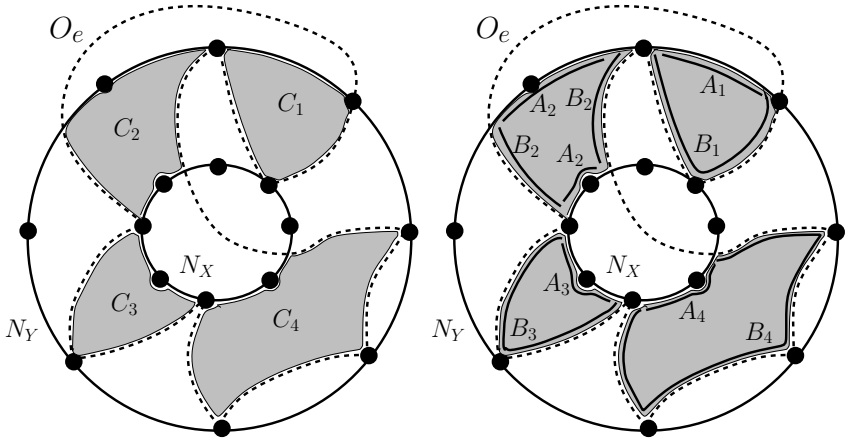


Figure 3.10: **Partial components and partial nooses.** The left diagram illustrates how O_e and N_X,N_Y form the partial components $C_1, \ldots, C_4$. Observe that $C_1, \ldots, C_4$ only intersect in vertices whereas O_e and N_X,N_Y do not have to. In the right diagram, each partial component is bounded by partial nooses. Only component C_2 has $|A_2| = |B_2| = 2$.

Proof. Assume, there is a component C_i with two partial noose $P_i^1, P_i^2 \in A_i \cap N_X$. Consider a walk along O_e. P_i^1 and P_i^2 are on the same side of that walk. Therefore, O_e must cross Δ_X twice, in contradiction to being a tight nose. For the second part of the proof, assume two components C_i and C_j with $|A_i| = |A_j| = 2$. Then there are four partial nooses of $B_i \cup B_j \subset O_e$ that must be connectible to a tight noose O_e. That is not possible without crossing Δ_X and Δ_Y more than once. □

See the right diagram of Figure 3.10 for an illustration.

In contrast to the first phase we encode the vertices for each component C_i of Proposition 3.6.3 separately. The connectors, the vertices that are in two components are encoded twice. A restriction to the encoding of the vertices in $O_e \cap (N_X \cup N_Y)$ is the consideration of the base encoding, for example a vertex with base value '[' or ']' cannot be assigned with '2' at any time.

We introduce new values for indicating a pairing with vertices of $\mathcal{X} = (G_e \setminus O_e) \cap (N_X \cup N_Y)$. Proposition 3.6.4 guarantees that we can differentiate between three types of partial components C_i. The ones without any vertex in $\mathcal{X}$ and the two that have properties 1 and 2. For all three cases every vertex of $V(C_i) \setminus O_e$ is contained in one path.

1. $V(C_i) \cap V(\mathcal{X}) = \varnothing$. (Component C_1 in Figure 3.10).

   - Every path has both endpoints in $V(C_i) \cap O_e$.

   - Every vertex of $V(C_i) \cap (N_X \cup N_Y)$ with base value '[' or ']' is not to be an inner vertex of a path.

- We use the same encoding as in phase 1.

2. $V(C_i) \cap V(\mathcal{X}) \neq \varnothing$ and $|A_i| = 1$. (Components C_3 and C_4 in Figure 3.10).

   - Every path has both endpoints in $V(C_i) \cap (O_e \cup \mathcal{X})$.

   - A vertex of $V(C_i) \cap O_e$ that is paired with the other endpoint w in $V(\mathcal{X})$ is encoded with the base value of w, that is, '[' or ']'. Since $|A_i| = 1$ and the Catalan structure is retained for the border vertices of C_i, it is possible to reconstruct the order π_{XY} in which the endpoints in $\mathcal{X}$ are. The base value '□' does not appear. We introduce 'S_X' and 'S_Y' for marking the connection to vertices in N_X and N_Y that have the base values 'S'. 'S_X' and 'S_Y' appear at most once.

   - For paths with both endpoints in $V(C_i) \cap O_e$ we use the same encoding as in phase 1.

3. $V(C_i) \cap V(\mathcal{X}) \neq \varnothing$ and $|A_i| = 2$. (Component C_2 in Figure 3.10).

   - Every path has both endpoints in $V(C_i) \cap (O_e \cup \mathcal{X})$.

   - As in the latter case we use the base encoding to encode the vertices of $V(C_i) \cap O_e$, too. Additionally, we introduce values '$]_X$','$[_X$','$]_Y$','$[_Y$' to mark each of the last two vertices in order π that are endpoint of a path with other endpoint in N_X and N_Y, respectively. These values are used only once in an unique C_i and hence do not play any role for the running time.

See Figure 3.11 for an example on the usage of encoding '$]_X$','$[_X$','$]_Y$','$[_Y$'.

Additional special cases.

- **Base value 'S'.**

  We omit here to consider the special cases that occur with base value 'S'. Recall that 'S' marks the fixed path $P_{i,j}$ and the beginning and the end of order π_{XY}. It is easy to see that the encoding determines whether a vertex of $V(C_i) \cap O_e$ is connected to a vertex before or after an endpoint of $P_{i,j}$. For example, suppose both endpoints of $P_{i,j}$ are in $\mathcal{X} \cup O_e$ and $P_{i,j} \subset C_i$. Then $P_{i,j}$ separates C_i into two parts which cannot be connected by a path, since neither $N_X \subset C_i$ nor $N_Y \subset C_i$.

- **Right encoding of connector.** Let c be a connector between two partial components C_i,C_j. The two values of c must combine to the correct base value. If base value of c is not '□', at least one of the two values in C_i or C_j must be '0' and none is '2'.

- **Path through several components.** For every component C_i, a vertex v of the tight noose $O_e \cap C_i$ can be paired to a connector with base value '□'. Hence, v can be an endpoint of a path with other endpoint in $\mathcal{X}$ in another component C_j. In this case assign v with the corresponding base value.

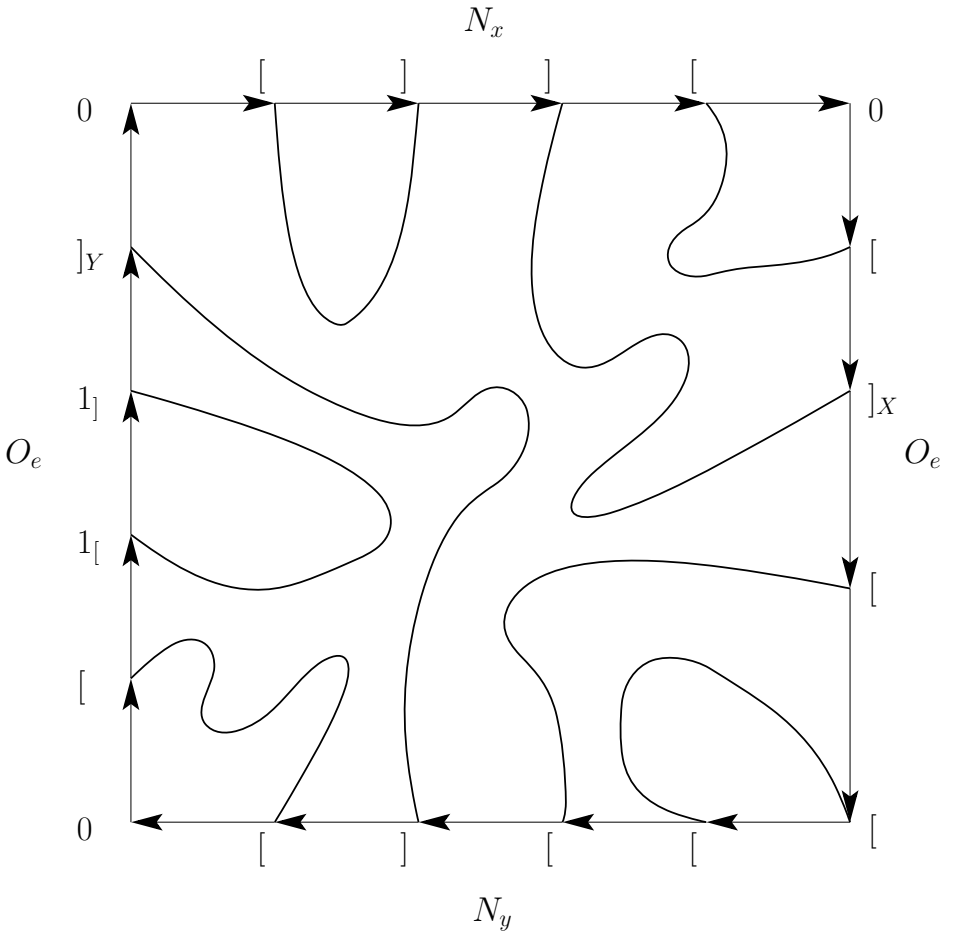


Figure 3.11: **Usage of '$]_X$','$[_X$','$]_Y$','$[_Y$'.** The diagram shows the partial component that is bounded by four partial nooses. The vertices are clockwise ordered beginning in the upper left corner. '$]_X$' on the right partial noose of O_e marks the last vertex of O_e connected to N_X. '$]_Y$' on the left partial noose of O_e marks the last vertex connected to N_Y.

Processing middle sets. The middle sets are processed exactly as described in the first phase. In the first step every pair of states of two child edges e_L and e_R are updated to a state of the parent edge e_P. For a symbol of $\{[,], S_X, S_Y,]_X, [_X,]_Y, [_Y\}$ the numerical value is 1 and we form the vertex assignments as above. I.e., base values are treated exactly as '$1_[$' or '$1_]$'. With the only restriction if the assignments c_L and c_R of a vertex v both are base values. The base value in O_L must fit to the base value in O_R, i.e., if wlog $c_L(v)$ has value '[' then $c_R(v)$ must have ']', if wlog $c_L(v) = S_X$ then $c_R(v) = S_Y$. In the second step, we do not only check forbidden cycles, but consistency of the encoding regarding to the base values. With the help of an auxiliary graph consisting of G_L and G_R together with the partial components, we check the following:

1. The base values of c_L and c_R are connected respecting π_{XY}. For a vertex v of I encoded by '[' and ']' in O_L and O_R, respectively, it must hold that the endpoint with base value '[' must be in order π_{XY} before the endpoint with ']'.
2. The vertices of a partial component C_i of O_P that are paired to a vertex of $\mathcal{X} \cap C_i$ are assigned with the correct value of $\{[,], S_X, S_Y,]_X, [_X,]_Y, [_Y\}$. Note that new connector vertices are generated, which must be encoded component-wise.

Running time. When counting the number of states, omit values $\{S_X, S_Y,]_X, [_X,]_Y, [_Y\}$ since they are assigned to at most two vertices of O_L and O_R. Each connector is assigned with two values. The number of connectors can be in order size of a O_L and O_R, respectively. The values of the vertices in the symmetric difference sets L, R are transferred in time depending on the number of values $\{0, 1_[, 1_], 2, [,]\}$ and the number of valid encoding of the connectors. There are 25 ways of encoding a connector correctly. Apparently, if a vertex is a connector in O_L then it is not in O_R. To simplify matters, assume that $V(O_L)$ are connectors and $V(O_R)$ are not. Then the update time for L, R is $O(25^{|L \cap O_L|} 6^{|R \cap O_R|})$. There are 45 possible assignments for vertices in F to sum up to two. Thus, updating time for the F-set is $O(45^{|F|})$. With analogous calculations as before we get an overall running time $6^{0.5\ell} 25^{0.5\ell} 45^{0.5\ell} \approx 2^{6.36\ell}$.

Proposition 3.6.5. *Phase 3: The phase of dynamic programming with $\mathcal{X} \neq \varnothing$ and either $V(N_X) \subseteq \mathcal{X}$ or $V(N_Y) \subseteq \mathcal{X}$ or both takes time $O(2^{6.360\ell})$.*

In the last phase at least one of both cut-nooses N_X,N_Y are subsets of $V(G_e) \setminus O_e$. The difficulty is apparently to encode the endpoint of a path with one endpoint in such N_X, N_Y. We consider two cases:

1. The fixed path $P_{i,j}$ is crossing O_e.

   If both N_X and N_Y are in $V(G_e) \setminus O_e$ we assume the first vertex v assigned by 'S_X' in π and the other vertex w by 'S_Y'. Use encoding with '$[_X$','$]_X$' to mark the last vertex in the partial noose $(v, w) \subset O_e$ connected to N_X and '$[_Y$','$]_Y$' for the last vertex in the partial noose $(w, v) \subset O_e$ connected to N_Y. If wlog N_X crosses O_e we find a partial component C_i including N_Y. We mark two vertices

with '$[_X$','$]_X$','$[_Y$','$]_Y$' in the same way , no matter if there is one or two of 'S_X','S_Y' in $V(C_i) \cap O_e$.

2. $P_{i,j} \subset G_e \setminus O_e$.

   One vertex in O_e is marked '$[_X$' or '$]_X$' to be the last vertex in π connected to N_X and one vertex by '$[_Y$' or '$]_Y$' to be the last connected to N_X. If one of N_X,N_Y cross O_e we again find a partial component C_i can be encoded in that way.

Since the new values '$[_*$','$]_*$' appear only twice per middle set, they do not affect the running time. The algorithm works the same as in phase two, considering the two latter cases.

3.6.2 Proof of Lemma 3.4.8

As in the previous subsection, we preprocess the graph G' by deleting all vertices in $\mathfrak{N}$ that do not belong to any path in candidate $\mathbf{C}$.

Dynamic programming follows the same ideas as in the previous subsection without encoding and with slight changes caused by the extension of Propositions 3.6.3 and 3.6.4. Due to Proposition 1.2.5 we can have the case that G' consists of several components $G'_1, G'_2, \ldots$. We simply do dynamic programming for each component separately.

Consider subgraph G_e bordered by tight noose O_e and $\mathfrak{N}_e \subset \mathfrak{N}$ as the cut-nooses intersecting G_e:

Proposition 3.6.6. *The subgraph G_e is the union of partial components $C_1, \ldots, C_q$ ($q \geq 1$) such that for every i*

$C_i \cap (\bigcup_{r=1,r\neq i}^{q} C_r) \subseteq O_e \cap \mathfrak{N}_e$. *Furthermore, for every i, j, h, $C_i \cap C_j \cap C_h = \varnothing$.*

Proposition 3.6.7. *Each C_i is bordered by partial nooses of A_i of tight nooses of $\mathfrak{N}_e \setminus O_e$ and partial nooses of $B_i \subset O_e$ with $\bigcup_{i=1}^{q} A_i \cup B_i = G_e \cap (\mathfrak{N}_e \cup O_e)$ such that one of the following hold:*

1. *$|A_i| = |B_i| = 1$ with $A_i \subset N_X$ for a tight noose $N_X \in \mathfrak{N}_e$,*
2. *$|A_i| = |B_i| \leq 2g$ with each partial noose of A_i part of a different tight noose of $\mathfrak{N}_e$.*

For all partial components C_i, C_j with property 2: A_i contains at least one partial noose that is part of a cut-noose of $\mathfrak{N}_e$ that has no partial noose in A_j. There are at most $2g$ components with property 2 and $|\bigcup_{i=1}^{2g} A_i| \leq 2g$.

See Figure 3.12 for an illustration.

Because of the last statement of Proposition 3.6.7 the size of the union over all B_i is also bounded by $2g$. Hence, there are at most $4g^2$ vertices in O_e fixed endpoints of paths with other end in a cut-noose and $O((2g\,\mathbf{bw}(G'))^{4g^2}$ possibilities for assigning these values to $V(O_e)$.

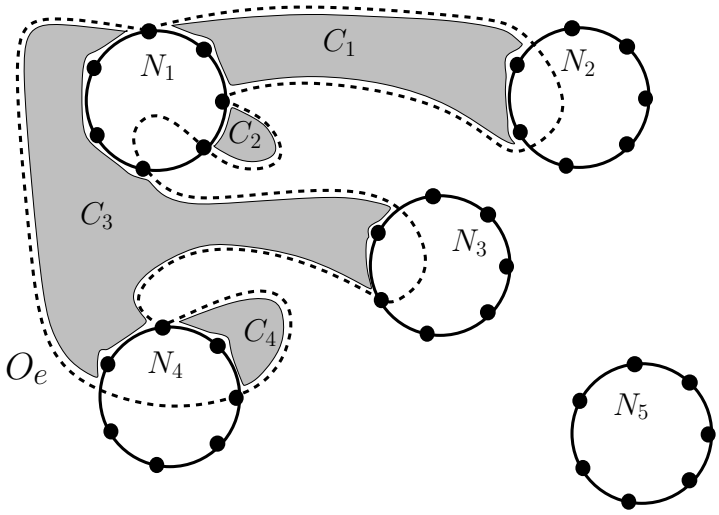


Figure 3.12: **Partial components with several cut-nooses.** The diagram shows how tight noose O_e intersects $\mathfrak{N}_e = \{N_1, \ldots, N_4\}$ and form the partial components $C_1, \ldots, C_4$. Observe that every partial noose of A_i $(1 \leq i \leq 4)$ is of a different cut-noose.

3.6.3 Proof of Theorem 3.5.2

Exit conditions for the algorithm

In this subsection, we give a lower bound on the branchwidth of the input graph G, that is, we give two exit conditions on which the algorithm terminates and fulfills the first part of Theorem 3.5.2, namely to give a certificate that G has large branchwidth. The following lemma follows from Theorem 4.1 of [84].

Lemma 3.6.8. *For any graph embeddable in a non planar surface, it holds that* $\mathbf{rep}(G) \leq \mathbf{bw}(G)$.

The function $f(H)$ in Step **2.c.ii** of the algorithm is defined by the following lemma that follows from [34, Lemmata 5,6, and 7].

Lemma 3.6.9. *Let G be an H-minor-free graph and let G_u^s as in Step* **2.a**. *Then, there exists a function $f(H)$ such that if $\mathbf{bw}(G_u^s) \geq f(H) \cdot w$, then G contains a $(w \times w)$-grid as a minor.*

The following lemma justifies the first terminating condition for the algorithm, depending on the value of $f(H)$ estimated in Lemma 3.6.9.

Lemma 3.6.10. *If in the x-th application of Step* **2.c.ii**, $|N| \geq 2^{x-1} f(H) \cdot w$, *then $G_u^s = G_u^{(1)}$ has branchwidth at least $f(H) \cdot w$.*

Proof. Let $N_1, \ldots, N_{x-1}$ be the nooses along which we cut the graphs $G_u^{(1)}, \ldots, G_u^{(x-1)}$ in Step **2.c.iv** toward creating $G_u^{(x)}$. We have that

$$\sum_{j=1,\ldots,x-1} |N_j| \leq \sum_{j=1,\ldots,x-1} 2^{j-1} f(H) \cdot w = (2^{x-1} - 1) f(H) \cdot w.$$

We also observe that G_u^{j-1} contains as a subgraph the graph taken from G_u^j if we remove one copy by each of its $|N_{j-1}|$ duplicated vertices. This implies that

$$\mathbf{bw}(G_u^{j-1}) \geq \mathbf{bw}(G_u^j) - |N_{j-1}|, j = 2, \ldots, x.$$

Inductively, we have

$$\mathbf{bw}(G_u^1) \geq \mathbf{bw}(G_u^x) - \sum_{j=1,\ldots,x-1} |N_j| \geq \mathbf{bw}(G_u^x) - (2^{x-1} - 1) f(H) \cdot w.$$

We set $N = N_x$. By Lemma 3.6.8, $\mathbf{bw}(G_u^x) \geq \mathbf{rep}(G_u^x)$ and $\mathbf{rep}(G_u^x) \geq |N_x| \geq 2^{x-1} f(H) \cdot w$. Thus, we conclude that $\mathbf{bw}(G_u^1) \geq f(H) \cdot w$. □

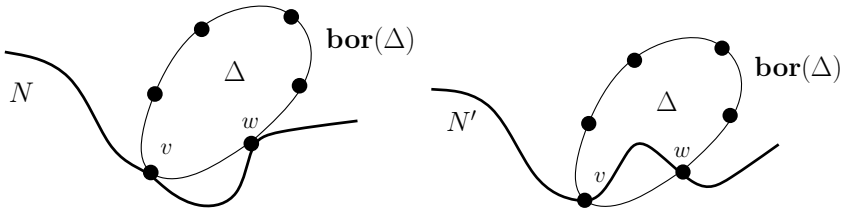


Figure 3.13: Re-routing a noose.

The following lemma justifies the first terminating condition for the algorithm, depending on the value of $f(H)$ and the genus γ_H.

Lemma 3.6.11. *If in Step* **2e**, $\mathbf{bw}(G_u^p) \geq 2^{\gamma_H - 1} f(H) \cdot w$, *then* $G_u^s = G_u^{(1)}$ *has branchwidth at least* $f(H) \cdot w$.

Proof. The proof is the same as the proof of Lemma 3.6.10 if we set $x = i$ and with deference that, in the end, we directly we have that $G_u^{(i)} = G_u^p$ has branchwidth at least $2^{i-1} f(H) \cdot w$. The result follows as the genus of G_u^s is bounded by γ_H and therefore $i \leq \gamma_H$. □

Enhancing the branch-decomposition of G_u^p

In the following lemma, we state how to enhance the branch-decomposition of G_u^p by the trunk decompositions of the vortices in order to obtain a branch-decomposition of G^p.

Lemma 3.6.12. *Let (T_u^p, τ_u^p) be a branch-decomposition of G_u^p and let (T^p, τ^p) be the branch-decomposition of G^p constructed in Step* **2.f**. *Then the width of (T^p, τ^p) is bounded by the width w of (T_u^p, τ_u^p) plus some constant that depends only on H.*

Proof. By the construction of (T^p, τ^p), for any $e \in E(T^p)$, $\partial E_e(G^p) \subseteq \mathbf{X}(\partial E_e(G_u^p))$. As the vertices of $\partial E_e(G_u^p)$ are the vertices of some tight noose N_e of $\mathbb{S}_0$, and this noose meets at most $r \leq h$ vortex disks we have that there are at most $2r \leq 2h$ vertices of $\partial E_e(G_u^p)$ that are members of some base sets B. Therefore, for any $e \in E(T^p)$, $|\partial E_e(G^p)| \leq w + 2h^2$. We conclude that the width of (T^p, τ^p) is at most $w + 2h^2$. □

Toward Catalan structure

The whole subsection is devoted to proving why the branch-decomposition we constructed for a smoothly almost-embeddable graph has the Catalan structure. With the following lemma, we can inductively show how we obtain a branch-decomposition with the Catalan structure for a smoothly almost-embeddable graph to a higher surfaces from the branch-decomposition of its planarized version.

Lemma 3.6.13. *Let G, G' be two almost-embeddable graphs created successively during Step* **2.c** *($i \geq 2$), let N be the noose along which G_u was cut toward constructing G'_u and G' and let N_1 and N_2 be the boundaries of the two holes of G'_u created after this splitting during Step* **2.c.iv**. *Let also (T', τ') be a branch-decomposition of G' and let (T, τ) be the branch-decomposition of G defined if $T = T'$ and $\tau = \tau' \circ \sigma$ where $\sigma : E(T) \to E(T')$ is the bijection pairing topologically equivalent edges in G and G'. For any $e \in E(T) = E(T')$, the following hold:*

a. $|\omega_G(e)| \leq |\omega_{G'}(\sigma(e))| + |\mathbf{X}(N)|$.

b. if $S \subseteq V(E_e)$, the number $|\mathbf{paths}_G(E_e, \partial E_e \cup S)|$ is bounded by

$$|\mathbf{paths}_{G'}(\sigma(E_e), \partial\sigma(E_e) \cup S \cup \mathbf{X}(N_1 \cap V(\sigma(E_e))) \cup \mathbf{X}(N_2 \cap V(\sigma(E_e))))|$$

Proof. To see $|\omega_G(e)| \leq |\omega_{G'}(e)| + |\mathbf{X}(N)|$, it is enough to observe that the identification of vertices in a graph may only add identified vertices in the border of an edge set and that any overlying set is a separator.

The second relation follows from the fact that any path in $G[E_e]$ connecting endpoints in $\partial E_e \cup S$ is the concatenation of a set of paths in $G'[E_e]$ connecting endpoints in $\partial E_e \cup S$ but also the vertices of cut-noose N_1, N_2 that are endpoints of E_e (along with their overlying sets). □

Fixing paths in smoothly h-almost embeddable graphs on the sphere. The remainder of this subsection is devoted to proving the following lemma:

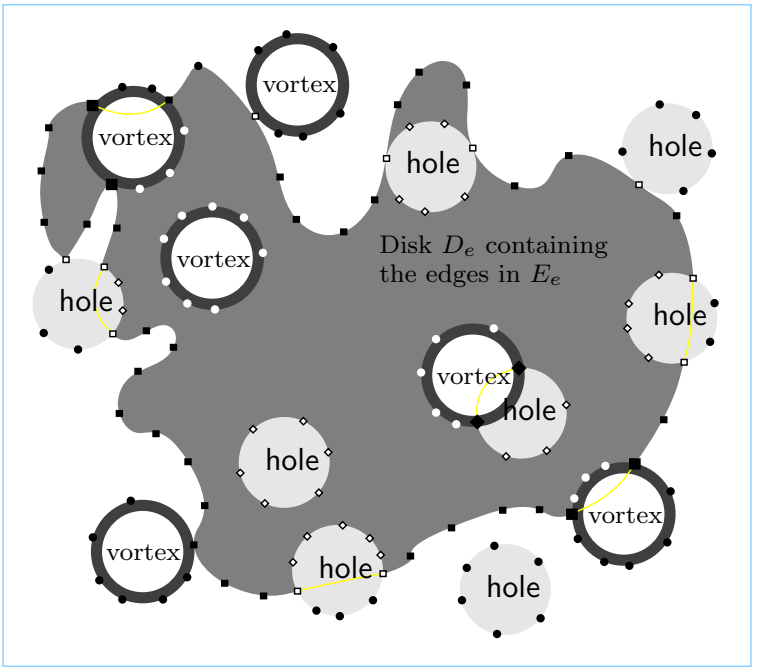


Figure 3.14: Vortices and holes around the disk Δ_e.

Lemma 3.6.14. *Let G^p be a smoothly $O_H(1)$-almost embeddable graph in the sphere and let $\Delta_1, \ldots, \Delta_r$ ($r = O_H(1)$) be disjoint closed disks (holes) of the sphere whose interior does not intersect the underlying graph G^p_u. Assume also that, if a vortex disk and a hole intersect, then they have common interior points. Let Δ_e be a closed disk of the sphere whose boundary is a tight noose touching G^p_u in vertex set ∂E_e. We denote as D_e the set containing all points on the boundary of the disks $\Delta_1, \ldots, \Delta_r$ that are endpoints of edges in $G^p_u \cap \Delta_e$ and as E_e the set of edges in $G^p \cap \Delta_e$. In G^p, let $\mathbf{X}(\partial E_e \cup D_e)$ be the overlying set of vertex set $\partial E_e \cup D_e$. If $|\mathbf{X}(\partial E_e \cup D_e)| = O_H(w)$, then*

$$\texttt{q-paths}_{G^p}(E_e, \mathbf{X}(\partial E_e \cup D_e)) = 2^{O_H(w)}.$$

In the following and for an easier estimation on $\texttt{q-paths}_{G^p}(E_e, \mathbf{X}(\partial E_e \cup D_e))$, we stepwise transform the graph in a way such that neither of the holes, the vortices and ∂E_e mutually intersect, by simultaneously nondecreasing the number of sets of paths.

Inverse edge contractions. From now on we will use the notation $\mathbf{V_e}$ for the vertex set $\mathbf{X}(\partial E_e \cup D_e)$.

The operation of *inverse edge contraction* is defined by duplicating a vertex v and connecting it to its duplicate v' by a new edge. However, we have that v maintains all its incident edges.

We say that two closed disks Δ_1 and Δ_2 *touch* if their interiors are disjoint and they have common points that are vertices. These vertices are called *touching vertices* of the

closed disks Δ_1 and Δ_2.

In order to simplify the structure of the planar embedding of G^p we will apply a series of inverse edge contractions to the touching vertices between the boundary of Δ_e and the vortex disks and holes.

Also, we assume that if we apply inverse edge contraction on a base vertex v of a vortex, v keeps all its incident edges and the duplicate of the respective boundary of a hole and of Δ_e has degree one. This creates a new graph G^{p*} that contains G^p as a minor and thus, each set of paths in G^p corresponds to a set of paths in G^{p*}.

We obtain the following:

Lemma 3.6.15. *Let $E_e, \mathbf{V_e}$ be as above. Then,*

$$\texttt{q-paths}_{G^p}(E_e, \mathbf{V_e}) \leq \texttt{q-paths}_{G^{p*}}(E_e^*, \mathbf{V_e^*}).$$

Where E_e^* and $\mathbf{V_e}^*$ are the enhanced sets in G^{p*}.

The red lines in the diagram in Figure 3.15 emphasize inverse edge contractions.

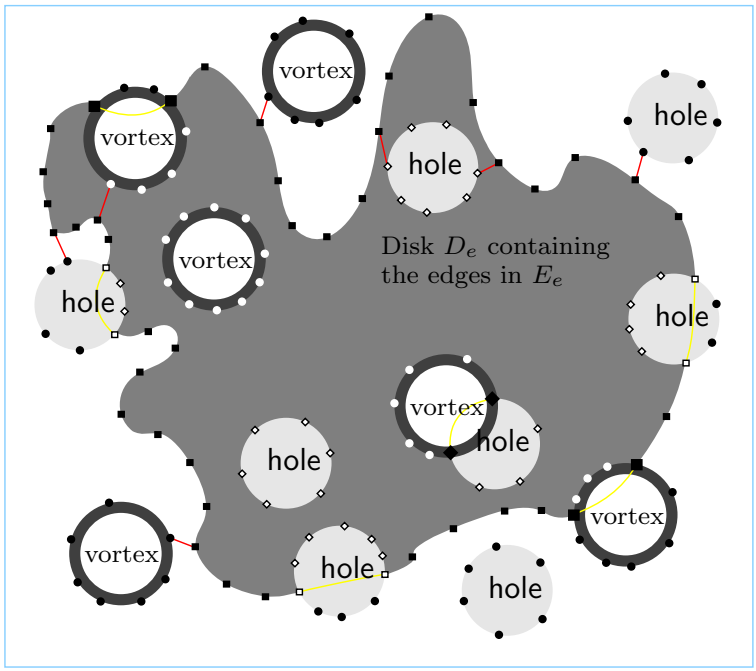


Figure 3.15: Vortices and holes not intersecting the disk Δ_e (first normalization).

Notice that each splitting creates duplicates some vertex of $\mathbf{V_e}$. Therefore,

Lemma 3.6.16. $|\mathbf{V_e}^*| \leq 2|\mathbf{V_e}|$.

On the left of Figure 3.16, we have the now resulting graph of Figure 3.15 where the gray part is $G^{p*}[E_e]$. On the right, we only emphasize $G^{p*}[E_e]$ as the part where the sets of $\mathbf{paths}_{G^{p*}}(E_e^*, \mathbf{V_e}^*)/\sim$ should be drawn.

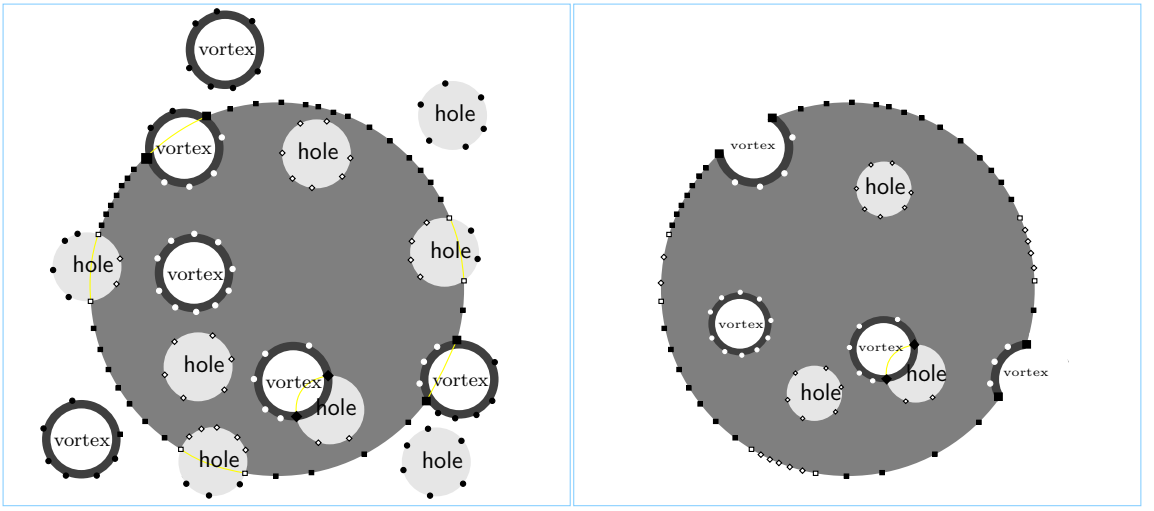


Figure 3.16: Another way to see Figure 3.15.

Vortex pattern. In order to have a more uniform image on how paths cross a vortex, we define the graph $R_{h,s}$ so that

$$V(R_{h,s}) = V_1 \cup \cdots \cup V_s \text{ with } |V_i| = h \text{ and}$$

$$E(R_{h,s}) = \{\{x_j, x_k\} \mid x_j, x_k \in V_i, 1 \leq j \neq k \leq h, 1 \leq i \leq s\} \cup$$
$$\{\{x_j, y_j\} \mid x_j \in V_{i-1} \wedge y_j \in V_i, 1 \leq j \leq h, 1 \leq i \leq s\}.$$

In $R_{h,s}$ we also distinguish a subset $S \subseteq V(R_{h,s})$ containing exactly one vertex from any V_i. We call the pair $(R_{h,s}, S)$ a (h, s)-vortex pattern. See Figure 3.17 for an example of a normalized vortex.

We now prove the following:

Lemma 3.6.17. *Any vortex of a h-almost embeddable graph with base set B is a minor of a (h, s)-vortex pattern $(R_{h,s}, S)$ where the minor operations map bijectively the vertices of S to the vertices in B in a way that the order of the vortex and the cyclic ordering of S induced by the indices of its elements is respected. $R_{h,s}$ has trunkwidth h.*

Proof. We show how any vortex with trunk decomposition $\mathcal{X} = (X_1, \ldots, X_{|B|})$ of width $< h$ and base set B, is a minor of some (h, s)-vortex pattern $(R_{h,s}, S)$ with $V(R_{h,s}) = V_1 \cup \cdots \cup V_s$ and $|V_i| = h (1 \leq i \leq s)$. Choose $s = |B|$ and set $S = B$. Start with the vertices in X_1: set $V_1 = X_1$ plus some additional vertices to make $|V_1| = h$ and make $G[V_1]$ complete. Iteratively, set $V_i = X_i \setminus X_{i-1}$. Apply inverse edge contraction for all vertices in $X_i \cap X_{i-1}$ and add the new vertices to V_i. Again, add additional vertices to make $|V_i| = h$. Make $G[V_i]$ complete and add all missing edges between V_i and V_{i-1} in order to obtain a matching.

□

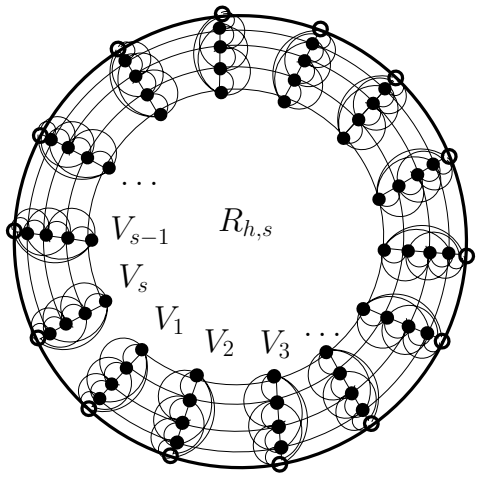
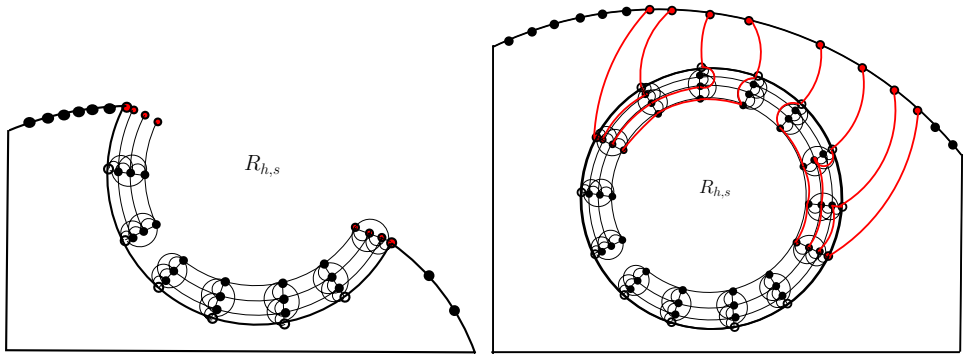


Figure 3.17: **Normalizing vortices.** An example of a $(6, 14)$-vortex pattern.

Using Lemma 3.6.17, we can replace G^p by a new graph $G^{p\prime}$ where any vortex of G^p is replaced by a suitably chosen $R_{h,s}$. The bijection of the lemma indicates where to stick the replacements to the underlying graph G^p_u. We adopt the same notions as for vortices. I.e, we denote S as base set consisting of base vertices, etc.. In the remainder of the paper we will refer to h-almost embeddable graphs as graphs with (h, s)-vortex pattern instead of vortices, unless we clearly state differently and we will use the term vortex for (h, s)-vortex pattern eventually.

Normalizing vortices. We can now apply one more transformation in order to have all vortex disks inside Δ_e and no vortices intersecting holes. Assume that for any (h, s)-vortex pattern we have that V_i and V_j are the two vertex sets of $R_{h,s} \cap \partial E_e$ (and $R_{h,s} \cap \mathbf{bor}(\Delta)$, Δ one of the holes $\Delta_1, \ldots, \Delta_r$, respectively.) We create $2h-2$ new vertex sets $V_i^1, \ldots, V_i^{h-1}$ and $V_j^1, \ldots, V_j^{h-1}$ to obtain a new $(h, s+2h-2)$-vortex pattern. We then apply inverse edge contraction on the base vertices of V_i and V_j and the new sets. This transformation is show in the next figure.

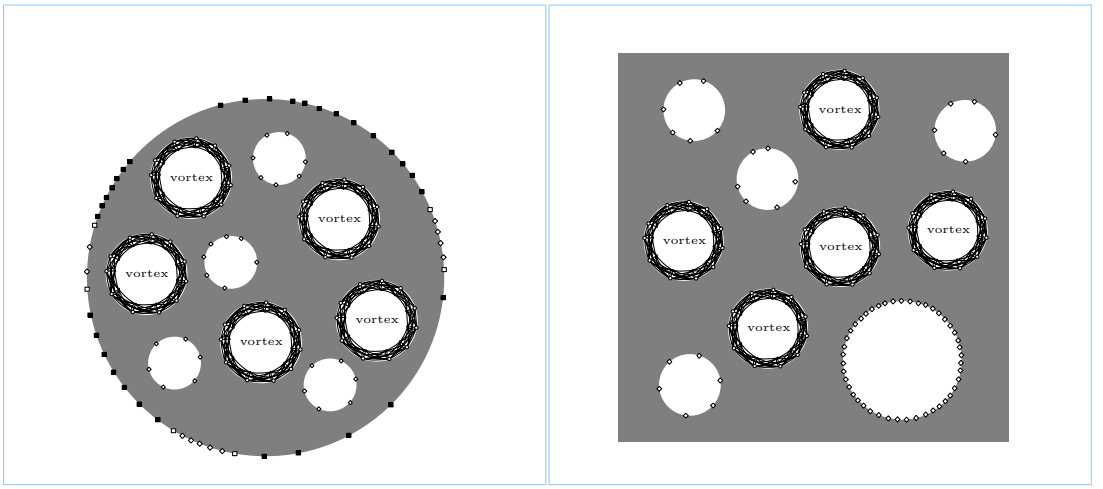


Again we can rename the graph before this new transformation G^p and the graph produced $G^{p\prime}$ and prove the following using the fact the later contains the former as a minor.

Lemma 3.6.18. *Let G^p and $G^{p\prime}$ be as above. Then*

$$\texttt{q-paths}_{G^p}(E_e, \mathbf{V_e}) \leq \texttt{q-paths}_{G^{p\prime}}(E'_e, \mathbf{V'_e}).$$

Final setting. To have an idea of how Δ_e looks like after the previous transformation, see the left part of the next figure. Clearly, the outer face can now be seen as a hole and we redraw the whole embedding in a sphere as indicated by the second part of the same figure. We will denote by $\Delta_{\partial E_e}$ the new disc in the graph embedding that is bounded by the union ∂E_e and some intersecting holes of $\Delta_1, \ldots, \Delta_r$.



In the current setting, we have a collection of holes in the sphere with ℓ vertices on their borders. We will now count $\texttt{q-paths}_{G^{p\prime}}(E'_e, \mathbf{V'_e})$ for the sets of vertex disjoint paths $\mathbf{paths}_{G^{p\prime}}(E'_e, \mathbf{V_e}')$ between these ℓ vertices.

Tree structure for fixing paths Before we are ready to prove Lemma 3.6.14, namely that

$$\texttt{q-paths}_{G^p}(E_e, \mathbf{V_e}) = 2^{O_H(w)},$$

we need the lemmas of Chapter 1.3.

We can apply Lemma 3.1.5 on h-almost-(n, r)-non-crossing matchings for Subsection 3.1.3 to our terminology:

We say two paths $P_1, P_2 \in \mathbf{paths}_{G^p}(E_e, \mathbf{V_e})$ *cross inside a (h, s)-vortex pattern* $(R_{h,s}, S)$ if there is a vertex set $V_i \in V(R_{h,s,})$ that is used by P_1 and P_2.

Each element of $\mathcal{P}^{O_H(1)}_{O_H(w),O_H(1)}$ is an equivalence class of paths $\mathbf{paths}_{G^p}(E_e, \mathbf{V_e})/\sim$ with $O_H(w)$ endpoints in $\mathbf{V_e}$ crossing inside $O_H(1)$ vortex patterns. Thus, we have proved Lemma 3.6.14, namely that

$$\texttt{q-paths}_{G^p}(E_e, \mathbf{V_e}) = |\mathcal{P}^{O_H(1)}_{O_H(w),O_H(1)}| = 2^{O_H(w)}.$$

Taming the apices

So far, we considered smoothly h-almost Σ- embeddable graphs G^s without apices. To include the apices, we enhance the branch-decomposition (T^s, τ^s) of G^s of width $O_H(w)$ so that each middle set contains at most all $O_H(1)$ apices. We construct an enhanced branch-decomposition (T, τ) of a h-almost Σ- embeddable graph G^a as follows: For every apex vertex α and for every neighbor v, choose an arbitrary edge e of T^s, such that $v \in \partial E_e$. Subdivide e and add a new edge to the new node and set $\tau(\{\alpha, v\})$ to be the new leaf. In this way, the enhanced branch-decomposition (T, τ) of G^a has width $O_H(w) + O_H(1)$. We obtain the following:

Lemma 3.6.19. *Given a h-almost embeddable graph G^a and its smoothly h-almost embeddable graph G^s after removing the apices with branch-decomposition (T^s, τ^s) of width $O_H(w)$. Then the enhanced branch-decomposition (T, τ) of G^a has width $O_H(w)$ and*

$$\texttt{q-paths}_{G^a}(E_e, \partial E_e) \leq w^{O_H(1)} \cdot \texttt{q-paths}_{G^s}(E^s_e, \partial E^s_e).$$

Proof. For an edge $e \in T$, we observe: Any path in $G^a[E_e]$ through an apex vertex α passes exactly two neighbors of α. If $G^a[E_e]$ does not contain any neighbor of any $\alpha \in G^a[E_e]$ then $\texttt{q-paths}_{G^a}(E_e, \partial E_e) \leq w^{O_H(1)} \cdot \texttt{q-paths}_{G^s}(E^s_e, \partial E^s_e)$. If $G^a[E_e]$ contains some neighbor, then α may be connected by a path to one or two vertices in ∂E^s_e. I.e., any apex vertex can only contribute to one path P in $G^a[E_e]$. Thus, for one α we count $O_H(w^2)$ different possible endpoint of P in ∂E^s_e, and for all $O_H(1)$ apices $w^{O_H(1)}$.

□

Taming the clique-sums

Given a graph an H-minor-free graph G . By Proposition 3.5.1, G can be decomposed in a tree-like way into several h-almost embeddable graphs by reversing the clique-sum operation. That is, we obtain a collection $\mathcal{C} = \{G^a_1, \ldots, G^a_n\}$ with each G^a_i being h-almost embeddable graphs with up to n (possibly intersecting) h-cliques that contributed to the clique-sum operation.

Lemma 3.6.20. *Given a graph G with branch-decomposition (T, τ). For any k-clique S in G, there are three adjacent edges e, f, g in T such that $S \subseteq \partial E_e \cup \partial E_f \cup \partial E_g$.*

Proof. Say, for node t incident to above e, f, g, $\partial E_t = \partial E_e \cup \partial E_f \cup \partial E_g$. We will prove the lemma inductively. Let a 3-clique consist of the vertices u, v, w. In (T, τ), we consider path $P_u \in T$ to be the path between the leaves $\tau(\{u, v\})$ and $\tau(\{u, w\})$. By definition, $u \in \partial E_e$ for all $e \in P_u$. Let node $t \in P_u$ be an endpoint of the path in $T \setminus P_u$ with other endpoint $\tau(\{v, w\})$. Then, $\{u, v, w\} \subseteq \partial E_t$. For an $i \leq k$, let $S_i \subset S$ be an i-clique for which there is a $t \in T$ with $S_i \subseteq \partial E_t$. Let $T_i \subseteq T$ be the tree induced by the paths between the leaves corresponding to the edges of S_i. Let $z \in S \setminus S_i$ and $T_z \subseteq T$ be the subtree induced by the paths connecting the leaves corresponding to edges between z and S_i. Then, we differ two cases: either $t \in T_i \cap T_z$ or there is a path in T_i connecting t and the closest node t_z in T_z. In the first case, under the assumption that $S_i \subseteq \partial E_t$ we obtain that $S_i \cup \{z\} \subseteq \partial E_t$. In the second case, since $S_i \subseteq \partial E_t$ and each vertices of S_i is endpoint of some edge in a leaf of T_z, we get that $S_i \subseteq \partial E_{t_z}$. By definition $z \in \partial E_{t_z}$ and we are done.

□

We define the node t incident to above edges $e, f, g \in T$ as a *k-clique-node*.

Since for any edge $e \in T$ for a branch-decomposition (T, τ), the vertex set ∂E_e separates the graph into two parts, we obtain the following lemma:

Lemma 3.6.21. *Given a graph G and a branch-decomposition (T, τ). For any edge $e \in T$ if $\texttt{q-paths}_G(E_e, \partial E_e) \leq q$ and $\texttt{q-paths}_G(\overline{E_e}, \partial\overline{E_e}) \leq q$ then $\texttt{q-paths}_G(E(G), \partial E_e) \leq q^2$. For any three adjacent edges e_1, e_2, and $e_3 \in T$, if $\texttt{q-paths}_G(E_{e_i}, \partial E_{e_i}) \leq q$ and $\texttt{q-paths}_G(\overline{E_{e_i}}, \partial\overline{E_{e_i}}) \leq q$ for $i = 1, 2, 3$ then $\texttt{q-paths}_G(E(G), \bigcup_{i=1,2,3} \partial E_{e_i}) \leq q^3$.*

We now show how to construct the branch-decomposition of a h-clique-sum by connecting the branch-decompositions of the two clique-sum components at some h-clique-nodes that correspond to the involved h-clique: Let G_1^a and G_2^a be the two clique-sum components with the cliques $S_i \subseteq V(G_i^a), (i = 1, 2)$ together with the branch-decompositions (T_i^a, τ_i^a) and a h-clique-node t^i. Then, the branch-decomposition (T', τ') of the clique-sum G' is obtained by first subdividing an incident edge e_{t^i} and connecting the new nodes together. Secondly, remove each leaf l of T' that corresponds to an edge that has a parallel edge or is deleted in the clique-sum operation, and finally contract an incident edge in T' of each degree-two node.

Lemma 3.6.22. *Let G_1^a and G_2^a have branch-decompositions $(T_1^a, \tau_1^a), (T_2^a, \tau_2^a)$ with the maximum width w and for all edges $e \in T_1' \cup T_2'$ let $\texttt{q-paths}_{G'}(E_e, \partial E_e) \leq q$. The previous construction of the branch-decomposition (T', τ') of the h-clique-sum G' has width $\leq w + h$ and for all edges $e \in T'$ $\texttt{q-paths}_{G'}(E_e, \partial E_e) \leq q^2$.*

Proof. For all $e \in T'$, ∂E_e has the same cardinality as in $T_1^a \cup T_2^a$. Only for the edges e_{t^i}, we have that $\partial E_{e_{t^i}} \subseteq \partial E_{t^i}$. Hence, the width increases by at most h.

For any tree edge $e \in T'$, let L be a set of leaves in the subtree inducing E_e corresponding to the edges of the cliques S_i in E_e. Then, for all $\tau'(\{u, v\}) \in L$, both endpoints u, v are vertices in ∂E_e.

Let t^i be in the subtree inducing $\overline{E_e}$. Since in T_i^a, $\texttt{q-paths}_{G_i^a}(E_e, \partial E_e) \leq q$ and $\texttt{q-paths}_{G_i^a}(\overline{E_e}, \partial\overline{E_e}) \leq q$, and also $\texttt{q-paths}_{G_i^a}(E_e \cap E(S_i), \partial E_e \cap S_i) \leq |L|^{|L|}$, we have that in T $\texttt{q-paths}_{G'}(\overline{E_e}, \partial\overline{E_e}) \leq q \cdot |L|^{|L|} \leq q^2$ for $e \neq e_t$. With Lemma 3.6.21, and since $\partial E_{e_{t^i}} \subseteq \partial E_{t^i}$, we get $\texttt{q-paths}_{G'}(E_{e_{t^i}}, \partial E_{e_{t^i}}) \leq q^2$ and $\texttt{q-paths}_{G'}(\overline{E_{e_{t^i}}}, \partial\overline{E_{e_{t^i}}}) \leq q^2$.Deleting leaves from (T', τ') does neither increase the width nor increase the number of path collections. □

In this way, we construct the branch-decomposition (T, τ) of an H-minor-free graph G out of the branch-decompositions $(T_1^a, \tau_1^a), \ldots, (T_n^a, \tau_n^a)$ of ≤ 1.5 times the maximum width $O_H(w)$ of the h-almost embeddable graphs $G_1^a, \ldots, G_n^a$. Extending the same arguments of the previous proof, we get the following lemma.

Lemma 3.6.23. *Given above h-almost embeddable graphs $G_1^a, \ldots, G_n^a$ together with their branch-decompositions $(T_1^a, \tau_1^a), \ldots, (T_n^a, \tau_n^a)$ of maximum width $O_H(w)$ and a collection $\mathcal{S}$ of h-cliques, each in one of $G_1, \ldots, G_n$. Then, the new branch-decomposition (T, τ) of G has width $O_H(w)$ and for all edges $e \in T$, $\texttt{q-paths}_G(E_e, \partial E_e) \leq 2^{O_H(w)}$.*

Proof. (Sketch) Let L be the set of leaves for one branch-decompositions (T_j^a, τ_j^a) defined as above for all h-cliques of the h-clique-sum operation for G_j^a. Since the edges corresponding to L contribute already to the sets of $\texttt{q-paths}_G(E_e, \partial E_e)$ for all $e \in T$ with $e \neq e_{tj+1}$, we get that $\texttt{q-paths}_G(\overline{E_e}, \partial\overline{E_e}) \leq q^2$. With Lemma 3.6.21, and with $\partial E_{e_{tj+1}} \subseteq \partial E_{tj+1}$, $\texttt{q-paths}_G(E_{e_{tj+1}}, \partial E_{e_{tj+1}}) \leq q^3$.

□

4 Conclusion

4.1 Brief summary of results

4.1.1 Dynamic programming

Fast matrix multiplication. We established a combination of dynamic programming and fast matrix multiplication as an important tool for finding fast exact algorithms for NP-hard problems. Even though the currently best constant $\omega < 2.376$ of fast matrix multiplication is of rather theoretical interest, there exist some practical sub-cubic runtime algorithms that help improving the runtime for solving all mentioned problems. An interesting side-effect of our technique is that any improvement on the constant ω has a direct effect on the runtime behavior for solving the considered problems. E.g., for Planar Dominating Set: under the assumption that $\omega = 2$, we come to the point where the constant in the computation is 3 what equals the number of vertex states, which is the natural lower bound for dynamic programming. Currently, [96] have made some conjecture on an improvement for distance product, which would enable us to apply our approach to optimization problems with arbitrary weights.

It is easy to answer the question why our technique does not help for getting faster tree decomposition based algorithms. The answer lies in the different parameter; even though it can be practically an improvement, since tree decompositions can have the same structure as branch decompositions (semi-nice tree decompositions), fast matrix multiplication does not affect the theoretical worst case behavior. This is due to adjacent bags possibly overlapping in all vertices.

Geometric decompositions. For the planar case we answered the following question positively: can we change the structure of tree decompositions to compensate its disadvantage against sphere cut decompositions in matters of dynamic programming? We described how Jordan curves and separators in plane graphs influence each other and we got some tools for relating Jordan curves and tree-decompositions. Finally, we showed how to compute geometric tree-decompositions and stated their influence on dynamic programming approaches.

4.1.2 Graph structures

Planar graphs. We introduced a new algorithmic design technique based on geometric properties of branch decompositions. Our technique can be also applied to construct

$2^{O(\sqrt{n})} \cdot n^{O(1)}$-time algorithms for a variety of cycle, path, or tree subgraph problems in planar graphs like Hamiltonian Path, Longest Path, and Connected Dominating Set, and Steiner Tree amongst others.

Bounded-genus graphs. We introduced a new approach for solving edge-subset problems on graphs of bounded genus. With some modifications this generic approach can be used to design time $2^{O(\sqrt{n})}$ algorithms for many other problems including Σ-embedded Graph TSP (TSP with the shortest path metric of a Σ-embedded graph as the distance metric for TSP), Max Leaf Tree, and Steiner Tree.

Fixed minors. Similar results can be obtained for all problems examined in this section on H-minor-free graphs. Since property $\boxed{\Lambda}$ in Section 1.4 holds for minor/apex-contraction bidimensional parameters on H-minor-free/apex-minor-free graphs, we have an analogue of $\mathbf{q\text{-}paths}_G(E_e, \partial E_e)$ and we can design $2^{O(\sqrt{k})} \cdot n^{O(1)}$ step parameterized algorithms for all problems examined in this section for H-minor-free/ apex-minor-free graphs (here the hidden constant in the big-O notation in the exponent depends on the size on the excluded minor).

4.2 Ongoing research and open problems

Connectivity problems. We have proved in Chapter 3 that for planar graphs, bounded-genus graphs, and H-minor-free graphs problems like Hamiltonicity and Metric graph TSP possess subexponential algorithms. However, we miss a formal classification criterion (logical or combinatorial) for the problems that are amenable to this approach. In Section 1.3.2, we gave a formal description of edge subset problems with global properties, but we do not know at which properties our techniques collapse. An interesting question is to find or disprove such criterion for connectivity problems, in particular for NP-hard Planar Subgraph Isomorphism, that, given a fixed graph H, asks if H is isomorphic to a subgraph of a planar input graph. Eppstein [42] showed that Planar Subgraph Isomorphism problem with pattern of size k can be solved in time $2^{O(\sqrt{k}\log k)}n$. Can we get rid of the logarithmic factor in the exponent (maybe in exchange to a higher polynomial degree)? Or differently formulated, can we prove a classification criterion similar to that of ETH implying the non-existence of algorithms of runtime $2^{o(\sqrt{k}\log k)} \cdot n^{O(1)}$ for some problems?

Parameterized problems. A natural question appears: until what point property $\boxed{\Lambda}$ in Section 1.4 can be satisfied for contraction-closed parameters (assuming a suitable concept of bidimensionality)? As it was observed in [29], for some contraction-closed parameters, like Dominating Set , the branchwidth of an apex graph cannot be bounded by any function of their values. Consequently, apex-free graph classes draw a natural

combinatorial limit on the the above framework of obtaining subexponential parameterized algorithms for contraction-closed parameters. (On the other side, this is not the case for minor-closed parameters as indicated by Theorem 1.4.7.) However, it is still possible to cross the frontier of apex-minor-free graphs for the dominating set problem and some of its variants where subexponential parameterized algorithms exist, even for H-minor-free graphs, as shown in [31]. These algorithms are based on a combination of dynamic programming and the structural characterization of H-minor-free graphs from [85].

Semi-nice tree-decompositions. It is our hope that future research will improve further on the runtime analysis of dynamic programming on semi-nice tree-decompositions of Section 2.1.3. For example, in a Join node X with parent A and partition D, E, F the vertices in $Z = D \setminus A$ will be forgotten in the parent node A. At present it is not at all clear how to make us of this information, but it does raise the possibility of a Join update with faster runtime for these vertices and a focus on what we may call Z, D, E, F-partitions for Join nodes. Another issue is to get a better understanding of when pathwidth, treewidth or branchwidth is best, possibly relating this also to other graph parameters. Finally, it would be interesting, but probably difficult, to design algorithms that find semi-nice tree-decompositions whose Join partitions D, E, F are optimized to give the best worst-case runtime for a particular dynamic programming algorithm. Note that a non-optimal tree-decomposition could in fact be better than an optimal one. A focus on Join nodes with small Expensive sets should probably be the primary issue.

Fast matrix multiplication. An interesting question arises for dynamic programming in combination with fast matrix multiplication from Section 2.2. It comes to ones mind that the intersection set I is not considered at all for matrix multiplication. Is there anything to win for dynamic programming if we use 3-dimensional matrices as a data structure? That is, if we have the third dimension labeled with $S_e(I)$?

Geometric decompositions. A natural question to pose from Section 2.3 is: would it be possible to solve Planar Dominating Set in time $2.99^{\mathbf{tw}(\mathcal{T})} \cdot n^{O(1)}$ and Planar Independent Set in $1.99^{\mathbf{tw}(\mathcal{T})} \cdot n^{O(1)}$? Though we cannot give a positive answer yet, we have a formula that needs the property "well-balanced" separators in a geometric tree-decomposition $\mathcal{T}$: we assume that the three sets L, R, F are of equal cardinality for every three adjacent bags. Since $|L| + |R| + |F| \leq \mathbf{tw}$, we thus have that $|L|, |R|, |F| \leq \frac{\mathbf{tw}}{3}$. Applying the fast matrix multiplication method from 2.2 for example to Planar Independent Set, this leads to a $2^{\frac{\omega}{3}\,\mathbf{tw}(\mathcal{T})} \cdot n^{O(1)}$ algorithm, where $\omega < 2.376$. Does every planar graph have a geometric tree-decomposition with well-balanced separators?

Evaluation of subexponential algorithms for edge subset problems on planar graphs. The results of Cook & Seymour [22] on using branch decompositions to obtain high-

quality tours for (general) TSP show that branch decomposition based algorithms run much faster than their worst case time analysis would indicate. Together with the author's preliminary experience on the implementation of a similar algorithm technique for solving Planar Vertex Cover in [4], we conjecture that sc-decomposition based algorithms perform much faster in practice.

Bibliography

[1] J. Alber, *Exact algorithms for np-hard problems on networks: Design, analysis, and implementation*, PhD Thesis, Universität Tübingen, (2002).

[2] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier, *Fixed parameter algorithms for dominating set and related problems on planar graphs*, Algorithmica, 33 (2002), pp. 461–493.

[3] J. Alber, H. L. Bodlaender, H. Fernau, and R. Niedermeier, *Fixed parameter algorithms for planar dominating set and related problems*, in Proceedings of the seventh Scandinavian Workshop on Algorithm Theory (SWAT2000), vol. 1851 of LNCS, Springer, 2000, pp. 97–110.

[4] J. Alber, F. Dorn, and R. Niedermeier, *Experimental evaluation of a tree decomposition-based algorithm for vertex cover on planar graphs*, Discrete Applied Mathematics, 145 (2005), pp. 219–231.

[5] J. Alber, H. Fan, M. R. Fellows, H. Fernau, R. Niedermeier, F. A. Rosamond, and U. Stege, *Refined search tree technique for dominating set on planar graphs*, in Proceedings of the 26th International Symposium on the Mathematical Foundations of Computer Science (MFCS2001), vol. 2136 of LNCS, Springer, 2001, pp. 111–122.

[6] J. Alber, H. Fernau, and R. Niedermeier, *Parameterized complexity: exponential speed-up for planar graph problems*, J. Algorithms, 52 (2004), pp. 26–56.

[7] J. Alber and R. Niedermeier, *Improved tree decomposition based algorithms for domination-like problems*, in Proceedings of the fifth Latin American Theoretical Informatics Symposium (LATIN'02), vol. 2286 of LNCS, Springer, 2002, pp. 613–627.

[8] N. Alon, Z. Galil, and O. Margalit, *On the exponent of the all pairs shortest path problem*, Journal of Computer and System Sciences, 54 (1997), pp. 255–262.

[9] N. Alon, P. Seymour, and R. Thomas, *A separator theorem for nonplanar graphs*, J. Amer. Math. Soc., 3 (1990), pp. 801–808.

[10] N. Alon, R. Yuster, and U. Zwick, *Color-coding*, J. Assoc. Comput. Mach., 42 (1995), pp. 844–856.

[11] S. Arora, M. Grigni, D. Karger, P. Klein, and A. Woloszyn, *A polynomial-time approximation scheme for weighted planar graph TSP*, in Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1998), ACM, 1998, pp. 33–41.

[12] P. A. Bernstein and N. Goodman, *Power of natural semijoins*, SIAM Journal on Computing, 10 (1981), pp. 751–771.

[13] H. L. Bodlaender, *Treewidth: Algorithmic techniques and results*, in Proceedings of the 22nd International Symposium on the Mathematical Foundations of Computer Science (MFCS'97), vol. 1295 of LNCS, Springer, 1997, pp. 19–36.

[14] ——, *A tourist guide through treewidth*, Acta Cybernet., 11 (1993), pp. 1–21.

[15] ——, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25 (1996), pp. 1305–1317.

[16] H. L. Bodlaender and R. H. Möhring, *The pathwidth and treewidth of cographs*, SIAM Journal on Discrete Mathematics, 6 (1993), pp. 181–188.

[17] H. L. Bodlaender and D. M. Thilikos, *Constructive linear time algorithms for branchwidth*, in Proceedings of the 24th International Colloquium on Automata, Languages and programming (ICALP 1997), vol. 1256 of LNCS, Springer, 1997, pp. 627–637.

[18] V. Bouchitté, F. Mazoit, and I. Todinca, *Chordal embeddings of planar graphs*, Discrete Mathematics, 273 (2003), pp. 85–102.

[19] L. Cai and D. Juedes, *On the existence of subexponential parameterized algorithms*, J. Comput. System Sci., 67 (2003), pp. 789–807. Special issue on parameterized computation and complexity.

[20] A. Cayley, *A theorem on trees*, Quart J. Pure Appl. Math., 23 (1889), pp. 26–28.

[21] J. Chen, H. Fernau, I. A. Kanj, and G. Xia, *Parametric duality and kernelization: Lower bounds and upper bounds on kernel size*, in Proceedings of the 22nd International Symposium on Theoretical Aspects of Computer Science (STACS 2005), vol. 3403 of LNCS, Springer, 2005, pp. 269–280.

[22] W. Cook and P. Seymour, *Tour merging via branch-decomposition*, INFORMS Journal on Computing, 15 (2003), pp. 233–248.

[23] D. Coppersmith, *Rectangular matrix multiplication revisited*, Journal of Complexity, 13 (1997), pp. 42–49.

[24] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, Journal of Symbolic Computation, 9 (1990), pp. 251–280.

[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*, The MIT Press and McGraw-Hill Book Company, 2001.

[26] B. Courcelle, *Graph rewriting: An algebraic and logic approach*, in Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), 1990, pp. 193–242.

[27] A. Dawar, M. Grohe, and S. Kreutzer, *Locally excluding a minor*, preprint NI07001-LAA, Isaac Newton Institute of Mathematical Sciences, (2007).

[28] V. G. Deĭneko, B. Klinz, and G. J. Woeginger, *Exact algorithms for the Hamiltonian cycle problem in planar graphs*, Operations Research Letters, 34 (2006), pp. 269–274.

[29] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos, *Bidimensional parameters and local treewidth*, SIAM J. Discrete Math., 18 (2004), pp. 501–511.

[30] ——, *Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs*, ACM Trans. Algorithms, 1 (2005), pp. 33–47.

[31] ——, *Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs*, Journal of the ACM, 52 (2005), pp. 866–893.

[32] ——, *Fixed-parameter algorithms for the (k, r)-center in planar graphs and map graphs*,

in Proceedings of the 30th International Colloquium on Automata, Languages and programming (ICALP 2003), vol. 2719 of LNCS, Springer, 2003, pp. 829–844.

[33] E. D. Demaine and M. Hajiaghayi, *The bidimensionality theory and its algorithmic applications*, Computer Journal, to appear.

[34] ——, *Linearity of grid minors in treewidth with applications through bidimensionality*, Combinatorica, to appear.

[35] ——, *Bidimensionality: new connections between fpt algorithms and ptass*, in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005), ACM-SIAM, 2005, pp. 590–601.

[36] E. D. Demaine, M. T. Hajiaghayi, and K. Kawarabayashi, *Algorithmic graph minor theory: Decomposition, approximation, and coloring*, in Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), IEEE Computer Society, 2005, pp. 637–646.

[37] E. D. Demaine, M. T. Hajiaghayi, N. Nishimura, P. Ragde, and D. M. Thilikos, *Approximation algorithms for classes of graphs excluding single-crossing graphs as minors*, J. Comput. System Sci., 69 (2004), pp. 166–195.

[38] E. D. Demaine, M. T. Hajiaghayi, and D. M. Thilikos, *Exponential speedup of fixed-parameter algorithms for classes of graphs excluding single-crossing graphs as minors*, Algorithmica, 41 (2005), pp. 245–267.

[39] R. Diestel, *Graph theory, Third edition*, Springer-Verlag, Heidelberg, 2005.

[40] G. A. Dirac, *On rigid circuit graphs*, Abhandlungen Mathematisches Seminar Universitaet Hamburg, 25 (1961), pp. 71–76.

[41] R. G. Downey and M. R. Fellows, *Parameterized complexity*, Springer-Verlag, New York, 1999.

[42] D. Eppstein, *Subgraph isomorphism in planar graphs and related problems*, J. Graph Algorithms Appl., 3 (1999), pp. 1–27.

[43] U. Feige, M. Hajiaghayi, and J. R. Lee, *Improved approximation algorithms for minimum-weight vertex separators*, in Proceedings of the 37th annual ACM Symposium on Theory of computing (STOC 2005), ACM, 2005, pp. 563–572.

[44] H. Fernau and D. W. Juedes, *A geometric approach to parameterized algorithms for domination problems on planar graphs*, in Proceedings of the 29th International Symposium on the Mathematical Foundations of Computer Science (MFCS'04), vol. 3153 of LNCS, Springer, 2004, pp. 488–499.

[45] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Texts in Theoretical Computer Science. An EATCS Series, Springer-Verlag, Berlin, 2006.

[46] F. V. Fomin and D. M. Thilikos, *Dominating sets in planar graphs: branch-width and exponential speed-up*, in Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03), ACM, 2003, pp. 168–177.

[47] ——, *Fast parameterized algorithms for graphs on surfaces: Linear kernel and exponential speed-up*, in Proceedings of the 31st International Colloquium on Automata, Languages

and Programming (ICALP 2004), vol. 3142 of LNCS, Springer, 2004, pp. 581–592.

[48] ——, *A simple and fast approach for solving problems on planar graphs*, in Proceedings of the 21st International Symposium on Theoretical Aspects of Computer Science (STACS 2004), vol. 2996 of LNCS, Springer, 2004, pp. 56–67.

[49] ——, *Dominating sets in planar graphs: Branch-width and exponential speed-up*, SIAM J. Comput., 36 (2006), pp. 281–309.

[50] ——, *New upper bounds on the decomposability of planar graphs*, Journal of Graph Theory, 51 (2006), pp. 53–81.

[51] D. R. Fulkerson and O. A. Gross, *Incidence matrices and interval graphs*, Pacific Journal of Mathematics, 15 (1965), pp. 835–855.

[52] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

[53] F. Gavril, *The intersection graphs of subtrees in trees are exactly the chordal graphs*, Journal of Combinatorial Theory Series B, 16 (1974), pp. 47–56.

[54] J. R. Gilbert, J. P. Hutchinson, and R. E. Tarjan, *A separator theorem for graphs of bounded genus*, J. Algorithms, 5 (1984), pp. 391–407.

[55] M. Grohe, *Local tree-width, excluded minors, and approximation algorithms*, Combinatorica, 23 (2003), pp. 613–632.

[56] Q.-P. Gu and H. Tamaki, *Optimal branch-decomposition of planar graphs in $O(n^3)$ time*, in Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005), vol. 3580 of LNCS, Springer, 2005, pp. 373–384.

[57] G. Gutin, T. Kloks, C. M. Lee, and A. Yeo, *Kernels in planar digraphs*, J. Comput. System Sci., 71 (2005), pp. 174–184.

[58] P. Heggernes, *Minimal triangulations of graphs: A survey*, Discrete Mathematics, 306 (2006), pp. 297–317.

[59] P. Heggernes, J. A. Telle, and Y. Villanger, *Computing minimal triangulations in time $O(n^{\alpha}logn) = o(n^{2.376})$*, SIAM Journal on Discrete Mathematics, 19 (2005), pp. 900–913.

[60] C. W. Ho and R. C. T. Lee, *Counting clique trees and computing perfect elimination schemes in parallel*, Inf. Process. Lett., 31 (1989), pp. 61–68.

[61] R. Impagliazzo, R. Paturi, and F. Zane, *Which problems have strongly exponential complexity*, Journal of Computer and System Sciences, 63 (2001), pp. 512–530.

[62] A. Itai and M. Rodeh, *Finding a minimum circuit in a graph*, SIAM Journal on Computing, 7 (1978), pp. 413–423.

[63] I. Kanj and L. Perković, *Improved parameterized algorithms for planar dominating set*, in Proceedings of the 27th International Symposium Mathematical Foundations of Computer Science (MFCS 2002), vol. 2420 of LNCS, Springer, 2002, pp. 399–410.

[64] P. N. Klein, *A linear-time approximation scheme for planar weighted TSP*, in 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), IEEE Com-

puter Society, 2005, pp. 647–657.

[65] J. Kleinberg and E. Tardos, *Algorithm design*, Addison-Wesley, 2005.

[66] T. Kloks, *Treewidth*, vol. 842 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1994. Computations and approximations.

[67] T. Kloks, C. M. Lee, and J. Liu, *New algorithms for k-face cover, k-feedback vertex set, and k-disjoint cycles on plane and planar graphs*, in Proceedings of the 28th International Workshop on Graph-theoretic Concepts in Computer Science (WG 2002), vol. 2573 of LNCS, Springer, 2002, pp. 282–295.

[68] J. Kratochvil, *Perfect codes in general graphs*, (monograph) Academia Praha, Praha, 1991.

[69] D. Kratsch and J. Spinrad, *Between $O(nm)$ and $O(n^\alpha)$*, in Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03), ACM, 2003, pp. 158–167.

[70] G. Kreweras, *Sur les partition non croisées d'un circle*, Discrete Mathematics, 1 (1972), pp. 333–350.

[71] R. J. Lipton and R. E. Tarjan, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.

[72] ——, *Applications of a planar separator theorem*, SIAM J. Comput., 9 (1980), pp. 615–627.

[73] G. L. Miller, *Finding small simple cycle separators for 2-connected planar graphs*, Journal of Computer and System Science, 32 (1986), pp. 265–279.

[74] B. Mohar and C. Thomassen, *Graphs on surfaces*, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, 2001.

[75] R. Niedermeier, *Invitation to fixed-parameter algorithms*, vol. 31 of Oxford Lecture Series in Mathematics and its Applications, Oxford University Press, Oxford, 2006.

[76] C. H. Papadimitriou and M. Yannakakis, *On limited nondeterminism and the complexity of the v-c dimension*, J. Comput. Syst. Sci., 53 (1996), pp. 161–170.

[77] A. Parra and P. Scheffler, *Characterizations and algorithmic applications of chordal graph embeddings*, Discrete Applied Mathematics, 79 (1997), pp. 171–188.

[78] S. Parter, *The use of linear graphs in Gauss elimination*, SIAM Review, 3 (1961), pp. 119–130.

[79] B. Reed, *Treewidth and tangles, a new measure of connectivity and some applications*, Surveys in Combinatorics, 1997.

[80] N. Robertson and P. D. Seymour, *Graph minors X. Obstructions to tree-decomposition*, Journal of Combinatorial Theory Series B, 52 (1991), pp. 153–190.

[81] ——, *Graph minors. II. Algorithmic aspects of tree-width*, Journal of Algorithms, 7 (1986), pp. 309–322.

[82] ——, *Graph minors. VII. Disjoint paths on a surface*, J. Combin. Theory Ser. B, 45 (1988), pp. 212–254.

[83] ——, *Graph minors. X. Obstructions to tree-decomposition*, J. Combin. Theory Ser. B, 52 (1991), pp. 153–190.

[84] ——, *Graph minors. XI. Circuits on a surface*, J. Combin. Theory Ser. B, 60 (1994), pp. 72–106.

[85] ——, *Graph minors. XVI. Excluding a non-planar graph*, J. Combin. Theory Ser. B, 89 (2003), pp. 43–76.

[86] N. Robertson, P. D. Seymour, and R. Thomas, *Quickly excluding a planar graph*, J. Combin. Theory Ser. B, 62 (1994), pp. 323–348.

[87] D. Rose, R. E. Tarjan, and G. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM Journal on Computing, 5 (1976), pp. 146–160.

[88] R. Seidel, *On the all-pairs-shortest-path problem in unweighted undirected graphs*, Journal of Computer and System Sciences, 51 (1995), pp. 400–403.

[89] P. D. Seymour and R. Thomas, *Call routing and the ratcatcher*, Combinatorica, 14 (1994), pp. 217–241.

[90] A. Shoshan and U. Zwick, *All pairs shortest paths in undirected graphs with integer weights*, in Proceedings of the 40th Annual Symposium on Foundations of Computer Science, (FOCS '99), IEEE Computer Society, 1999, pp. 605–615.

[91] R. E. Tarjan and M. Yannakakis, *Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*, SIAM Journal on Computing, 13 (1984), pp. 566–579.

[92] J. A. Telle and A. Proskurowski, *Algorithms for vertex partitioning problems on partial k-trees*, SIAM J. Discrete Math, 10 (1997), pp. 529–550.

[93] C. Thomassen, *Embeddings of graphs with no short noncontractible cycles*, J. Combin. Theory Ser. B, 48 (1990), pp. 155–177.

[94] P. M. Vaidya, *Speeding-up linear programming using fast matrix multiplication*, in Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS 1989), IEEE Computer Society, 1989, pp. 332–337.

[95] J. van Leeuwen, *Graph algorithms*, MIT Press, Cambridge, MA, USA, 1990.

[96] V. Vassilevska and R. Williams, *Finding a maximum weight triangle in $n^{(3-\delta)}$ time, with applications*, in Proceedings of the 38th annual ACM Symposium on Theory of computing (STOC 2006), ACM, 2006, pp. 225–231.

[97] R. Williams, *A new algorithm for optimal constraint satisfaction and its implications*, in Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004), vol. 3142 of LNCS, Springer, 2004, pp. 1227–1237.

[98] ——, *A new algorithm for optimal 2-constraint satisfaction and its implications*, Theoretical Computer Science, 348 (2005), pp. 357–365.

[99] U. Zwick, *All pairs shortest paths using bridging sets and rectangular matrix multiplication*, Journal of the ACM, 49 (2002), pp. 289–317.

Index