Technische Universität München

Fakultät für Mathematik

Lehrstuhl für Angewandte Geometrie und Diskrete Mathematik

On the Minimum Bisection Problem in Tree-Like and Planar Graphs – Structural and Algorithmic Results

Tina Janne Schmidt

Vollständiger Abdruck der von der Fakultät für Mathematik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 27.12.2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Mathematik am 02.05.2017 angenommen.

Summary

This thesis studies the Minimum Bisection Problem in tree-like and planar graphs. The problem asks to partition the vertex set of a given graph into two classes of equal size while minimizing the width of the bisection, i. e., the number of edges between the classes. This optimization problem has many applications including in parallel computing and VLSI design. The problem is known to be NP-hard but, for graphs with bounded tree-width and grid graphs without holes as well as certain other graph classes, polynomial-time algorithms for computing a minimum bisection are known. Currently, the best approximation algorithm for the Minimum Bisection Problem achieves a ratio of $\mathcal{O}(\log n)$, where n denotes the number of vertices of the input graph. In the literature, no better approximation algorithm is known for planar graphs, and the question whether the Minimum Bisection Problem remains NP-hard when restricted to planar graphs is open.

It is known that bounded-degree trees and bounded-degree planar graphs on n vertices admit bisections of width $\mathcal{O}(\log n)$ and $\mathcal{O}(\sqrt{n})$, respectively. One of the aims of this thesis is to investigate graphs for which these bounds are tight up to a constant factor. It is shown that such trees have diameter $\mathcal{O}\left(\frac{n}{\log n}\right)$ and that such planar graphs have tree-width $\Omega(\sqrt{n})$ and, thus, contain a $k \times k$ grid with $k = \Omega(\sqrt{n})$ as a minor. Both results are proved by constructively establishing an upper bound for the width of a minimum bisection that depends on the diameter and the tree-width, respectively.

Another aim is to employ these constructive methods for the development of algorithms. In the case of trees, a bisection whose width is bounded in terms of the diameter and the maximum degree can be computed in linear time. This algorithm is generalized to tree-like graphs such that, if a tree decomposition $(T, \mathcal{X})$ of a graph G is given as input, a bisection in G, whose width is bounded in terms of the maximum degree of G, the relative weight of a heaviest path in $(T, \mathcal{X})$, and the width of $(T, \mathcal{X})$, is computed in time proportional to the encoding length of $(T, \mathcal{X})$.

For planar graphs, an algorithm constructing a bisection by using separators is proposed. A planar graph G on n vertices has large minimum bisection width if every bisection in G has width $\Omega\left(\sqrt{n}\right)$. As mentioned above, every bounded-degree planar graph on n vertices with a large minimum bisection width contains a $k \times k$ grid with $k = \Omega(\sqrt{n})$ as a minor. It is easy to see that the reverse implication does not hold. It is shown here that, if a graph G contains a large grid as a minor, which is spread homogeneously through the graph G, then G must have large minimum bisection width.

Finally, the more general problem Minimum k-Section is studied. This problem asks to partition the vertex set of a given graph into k classes of equal size while minimizing the width of the k-section. If k is part of the input and $c < 1$ is an arbitrary constant, then, in the case of trees on n vertices, it is NP-hard to approximate an optimal solution for the Minimum k-Section Problem within a ratio of n^c. Here, for bounded-degree trees with linear diameter, a polynomial-time algorithm approximating an optimal solution for the Minimum k-Section Problem within a constant ratio is presented. Afterwards, this algorithm is generalized to tree-like graphs such that, when given a tree decomposition $(T, \mathcal{X})$ of a graph G as input, a k-section in G, whose width is bounded in terms of k, the maximum degree of G, the relative weight of a heaviest path in $(T, \mathcal{X})$, and the width of $(T, \mathcal{X})$, is computed in polynomial time.

Acknowledgments

First and foremost, I would like to thank Prof. Anusch Taraz for his support and encouragement during my entire studies. I greatly appreciate that he always tried to make time for discussing problems and helped to match my personal plans with my work. Second, I would like to thank Prof. Cristina G. Fernandes for her constant interest in my work and welcoming me warmly as a visitor for so many times, not only at the university but also during her free time so that São Paulo almost feels like a second home by now. The discussions with both of them have shaped this thesis, my work, and my life in many aspects. Thank you for sharing your extensive knowledge with me, all the inspiration, your patience, and the nice times together!

The TopMath program of TUM Graduate School and the ENB provided me with the fruitful opportunity to start working on research questions under the supervision of Prof. Anusch Taraz at the end of my Bachelor's studies and also financially supported several trips to conferences. Concerning financial support, I gratefully acknowledge a scholarship of the Evangelische Studienwerk Villigst, which provided a great amount of independence and made it possible to solely focus on research for three years without worrying about any teaching duties. Most of the work presented in this thesis has been done during this time. In particular, I am thankful for supporting a long term visit in São Paulo to work with Prof. Cristina G. Fernandes. Moreover, I appreciate the opportunities for further visits in São Paulo realized by a PROBRAL CAPES/DAAD Project.

Furthermore, my sincere thanks go to the research groups at Technische Universität München, Technische Universität Hamburg, and Universidade de São Paulo, who always provided a great working environment, room for stimulating discussions, as well as enjoyable lunch and coffee breaks. In particular, I want to thank Dennis Clemens, Alexander Haupt, Carl Georg Heise, Andrea Jiménez, Guilherme Mota, Damian Reding, and Katharina Zahnweh for reading parts of this thesis and for many helpful comments. Staying in São Paulo was much easier and definitely a lot more fun with the help of Hiệp Hàn, Andrea Jiménez, Giulia Maesaka, Guilherme Mota, Mariana L. Simões, and Chandu Thatte.

Last but not least, I wholeheartedly appreciate Oliver Meister's endeavor during the past years for traveling with me before or after conferences, for visiting me in São Paulo and bringing all kind of ingredients to bake German Christmas cookies, as well as for reminding me to take breaks from thinking about math and filling these breaks with lots of fun.

Table of Contents

Chapter 1

Introduction

Fair sharing is hard. This applies not only to everyday situations, but also when splitting the vertex set of a given graph into two classes of equal size while minimizing the number of edges between the classes. Solving this problem, which is called the Minimum Bisection Problem, is NP-hard while the Minimum Cut Problem, that does not require the classes to have equal size, can be solved in polynomial time.

First, the Minimum Bisection Problem and the Minimum k-Section Problem are introduced in Section 1.1.1 and Section 1.1.2, where also a selection of previous known results for both problems is presented. Related problems and applications are quickly summarized in Section 1.1.3. Afterwards, Section 1.2 gives an overview on the results presented in this thesis. Furthermore, Section 1.3 discusses connections between the results and explains where, in the following chapters, their proofs can be found.

1.1 Minimum Bisection and Related Problems

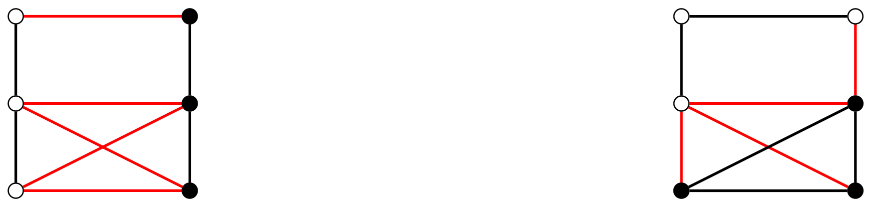
1.1.1 Minimum Bisection

A *bisection* (B, W) in a graph $G = (V, E)$ is a partition of its vertex set into two sets B and W, called the *black and the white set*, of sizes differing by at most one. An edge $\{x, y\}$ of G is *cut* by the bisection (B, W) if $x \in B$ and $y \in W$ or vice versa. The number of edges cut by the bisection (B, W) is called the *width* of the bisection and is denoted by $e_G(B, W)$. A *minimum bisection* in the graph G is a bisection (B^*, W^*) with

$$e_G(B^*, W^*) = \min\{e_G(B, W) \colon (B, W) \text{ is a bisection in } G\} =: \mathrm{MinBis}(G),$$

see Figure 1.1 for an example. Determining a bisection of minimum width is a famous optimization problem that is known to be NP-hard since 1976, see Theorem 1.3 in [GJS76]. In the following, we denote this problem by Minimum Bisection.

One way to deal with the hardness of a problem is to restrict the class of considered graphs. For example, when restricting the Minimum Bisection Problem to trees, it becomes solvable in polynomial time. Indeed, there is an algorithm that computes in $\mathcal{O}(n^3)$ time a minimum bisection in a tree on n vertices, see Theorem 4.3 in [Jan+05]. This algorithm by Jansen et al. relies on dynamic programming and can also be applied to *tree-like* graphs, i. e., graphs of constant tree-width. Roughly speaking, the *tree-width* of a graph G measures how tree-like G is. For example, trees have tree-width 1, cycles and cacti, i. e., graphs



a) A bisection (B, W) in G with $e_G(B, W) = 5$. **b)** A minimum bisection in G.

Figure 1.1: A graph G and two bisections (B, W). In both parts, the vertices in B are colored black and the vertices in W are colored white. Each edge of G that is cut by (B, W) is colored red.

where each edge is contained in at most one cycle, have tree-width 2, and a complete graph on n vertices has tree-width $n-1$. Returning to the algorithm in [Jan+05], when a tree decomposition of width t of a graph on n vertices is provided as input, then the algorithm computes a minimum bisection in $\mathcal{O}(2^t n^3)$ time. Using the algorithm in [Bod96] to compute a tree decomposition, the Minimum Bisection Problem becomes polynomially tractable for graphs of constant tree-width. As there are planar graphs on n vertices that do not allow a tree decomposition of width less than $\sqrt{n}$, the algorithm presented in [Jan+05] to compute a minimum bisection does not run in polynomial time for all planar graphs. In fact, it is open whether the Minimum Bisection Problem remains NP-hard when restricted to planar graphs. Díaz and Mertzios [DM14] believe that this is the case, since planar graphs and unit disk graphs often behave similarly with respect to computational complexity of optimization problems and they showed that the Minimum Bisection Problem restricted to unit disk graphs is NP-hard. A graph is a *unit disk graph* if its vertices can be mapped to points in the plane such that two vertices are adjacent if and only if the corresponding points have distance at most one. Also, Papadimitriou and Sideri [PS96] conjecture that the Minimum Bisection Problem remains NP-hard when restricted to planar graphs. They study *grid graphs*, which are finite induced subgraphs of the infinite grid. Consider the canonical embedding of a grid graph G. A *hole* of G is a face other than the infinite face whose boundary is not a cycle of length four. Papadimitriou and Sideri show that a minimum bisection in a grid graph on n vertices without holes can be computed in $\mathcal{O}(n^5)$ time and this approach can be generalized to run in $\mathcal{O}(n^{5+2h})$ time when applied to a grid graph on n vertices with h holes. Furthermore, they show that the Minimum Bisection Problem in planar graphs can be reduced to the Minimum Bisection Problem in grid graphs with an arbitrary number of holes. The algorithm for grid graphs on n vertices without holes has been improved to run in $\mathcal{O}(n^4)$ time by Feldmann and Widmayer [FW15]. Moreover, Bui et al. [Bui+87] showed that, for any fixed integer $d \geq 3$, the Minimum Bisection Problem remains NP-hard when restricted to d-regular graphs, i. e., for graphs $G = (V, E)$ where each vertex $v \in V$ satisfies $\deg(v) = d$. This immediately implies that the Minimum Bisection Problem restricted to graphs with maximum degree 3 is NP-hard.

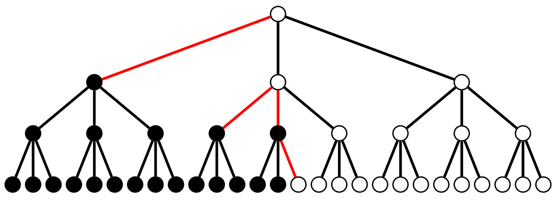
Another way to deal with the hardness of a problem is to study approximations. Roughly speaking, the idea is to compute a bisection that cuts few edges but might not be a minimum bisection. An algorithm is an α-approximation for the Minimum Bisection Problem if it computes a bisection (B, W) in the input graph G with $e_G(B, W) \leq \alpha \operatorname{MinBis}(G)$ in polynomial time. Currently, the best known approximation algorithm for the Minimum Bisection Problem is the $\mathcal{O}(\log n)$-approximation for arbitrary graphs on n vertices due to Räcke [Räc08]. Nothing better has been established for planar graphs, but an $\mathcal{O}(\log n)$-approximation for planar graphs on n vertices had been known before the result of Räcke, see [FK02]. When the minimum degree of the considered graph is linear, a *polynomial-time approximation scheme* for the Minimum Bisection Problem is known [AKK99], that is, for any fixed $\varepsilon > 0$, there is a polynomial-time $(1 + \varepsilon)$-approximation for graphs $G = (V, E)$ on n vertices, that satisfy $\deg(v) = \Omega(n)$ for all $v \in V$.

Consider a graph G that allows a bisection of constant width. Then, one can compute a minimum bisection in G in polynomial time by brute-force or, more precisely, by trying all possibilities to remove a constant number of edges from G and form a set of half of its vertices from the resulting components. The decision version of the Minimum Bisection Problem is to decide for an input graph G and input parameter e whether the graph G allows a bisection of width at most e. Cygan et al. [Cyg+14] describe an algorithm that solves this question in $\mathcal{O}\left(2^{\mathcal{O}(e^3)}n^3\log^3 n\right)$ time when given a graph G on n vertices and an integer e, which shows that the decision version of the Minimum Bisection Problem is *fixed parameter tractable*. Being fixed parameter tractable means that there is an arbitrary function f such that there is an algorithm for the decision version of the Minimum Bisection Problem that runs in time $\mathcal{O}(f(e)n^{\mathcal{O}(1)})$, i. e., polynomial in n but with arbitrary dependence on the parameter e, when given a graph G on n vertices and an integer e. Hence, when a graph G on n vertices with $\mathrm{MinBis}(G) = \mathcal{O}(\sqrt[3]{\log(n)})$ is considered, then $\mathrm{MinBis}(G)$ can be determined in polynomial time. Note that the brute-force approach results in an algorithm running in $\Omega\left(\binom{|E|}{e}\right)$ time for a graph $G = (V, E)$ and an integer e, and does not suffice to show that the decision version of the Minimum Bisection Problem is fixed-parameter tractable.

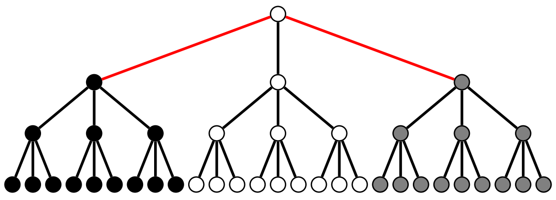
What upper bounds on the width of a minimum bisection for certain graph classes are known? Any graph $G = (V, E)$ with $n := |V|$ even satisfies $\mathrm{MinBis}(G) \le \frac{1}{2}|E|\frac{n}{n-1}$. Indeed, when a set $B \subseteq V$ of size $\frac{1}{2}n$ is chosen at random, then each edge in E is cut with probability $2 \cdot \frac{\frac{n}{2}}{n} \cdot \frac{\frac{n}{2}}{n-1} = \frac{1}{2} \cdot \frac{n}{n-1}$, so the expected width of the bisection (B, W) with $W := V \setminus B$ is $\frac{1}{2}|E|\frac{n}{n-1}$ and, hence, there is a bisection of width at most $\frac{1}{2}|E|\frac{n}{n-1}$ in G. The complete graph K_n with n even shows that this bound is tight, as every bisection (B, W) in K_n cuts $\frac{1}{4}n^2$ edges and $\frac{1}{2}|E(K_n)|\frac{n}{n-1} = \frac{1}{4} \cdot n(n-1) \cdot \frac{n}{n-1} = \frac{1}{4}n^2$. Since every tree $T = (V, E)$ satisfies $|E| = |V| - 1$, this implies that $\mathrm{MinBis}(T) \le \frac{1}{2}n$ for every tree T on n vertices with n even. The star $K_{1,n-1}$ on n vertices shows that this bound is tight.

Furthermore, consider a tree T on n vertices with maximum degree Δ_0. One can show that owing to the existence of a *separating vertex*, i. e., a vertex whose removal leaves no component of size greater than $\frac{1}{2}n$, a bisection of width at most $\Delta_0 \log_2(n)$ in T can be constructed and, hence, $\mathrm{MinBis}(T) \le \Delta_0 \cdot \log_2(n)$, see e. g. Corollary 4.9 in Chapter 4 where a slightly different method is used to derive the same bound. It is easy to see that a bisection satisfying this bound can be computed in $\mathcal{O}(n)$ time. Furthermore, the bound is tight up to a constant factor, because a perfect ternary tree T_h of height h satisfies $\mathrm{MinBis}(T_h) \ge h - \log_3(h)$, see Theorem 4.11 in [Sch13]. The method can be generalized to planar graphs by using planar separators as in [LT79] to obtain $\mathrm{MinBis}(G) = \mathcal{O}(\Delta_0\sqrt{n})$ for planar graphs G on n vertices with maximum degree Δ_0, see also Theorem 6.2 in [Jan+05]. Moreover, the bound for trees can be generalized to tree-like graphs by using tree decompositions to obtain $\mathrm{MinBis}(G) \le \Delta_0(\mathrm{tw}(G) + 1)\log_2(n)$ for every graph G on n vertices, where $\mathrm{tw}(G)$ denotes the tree-width of G. This method will also be used in Section 3.1 to construct bisections, where a proof for the bound for planar graphs and tree-like graphs is presented.

Lower bounds are more difficult to derive than upper bounds and only few are known. One example is the spectral bound $\mathrm{MinBis}(G) \ge \frac{1}{4}\lambda_2 n$ for graphs G on n vertices with n even, see Proposition 2.1 in [Moh92], where λ_2 denotes the second smallest eigenvalue of the Laplacian of G. The *Laplacian* of a graph G with $V(G) = [n]$ for some integer n is the matrix obtained from a diagonal matrix, whose i^{th} entry on the diagonal is $\deg_G(i)$, by subtracting the adjacency matrix of G. Chapter 1.9.1 in [Lei92] introduces another way to obtain a lower bound on the minimum bisection width in connected graphs. The idea is the following. Consider a connected graph G on n vertices and a bijection of the vertex set of K_n to the vertices of G as well as a function that maps each edge of K_n to a path joining the corresponding vertices of G. The *congestion* of an edge e is defined as the number of such paths that use e and the congestion C of the embedding of K_n into G is defined to be the maximum congestion among all edges of G. Then, $\mathrm{MinBis}(G) \ge \frac{1}{4C}n^2$ if n is even, and $\mathrm{MinBis}(G) \ge \frac{1}{4C}(n^2 - 1)$ if n is odd.



a) A minimum bisection (B, W), which has width 4. The vertices in B are colored black and the vertices in W are colored white.



b) A minimum 3-section (B_1, B_2, B_3), which has width 2. The vertices are colored black, white, and gray to indicate the set B_ℓ to which they belong.

Figure 1.2: Different k-sections of a perfect ternary tree. A perfect ternary tree is an example of a tree for which $\mathrm{MinSec}_k(T)$ does not increase monotonically as k increases.

1.1.2 Minimum k-Section

The concept of a minimum bisection can be generalized to splitting the vertex set of a graph into an arbitrary number of sets. For an integer $k \geq 2$, a *k-section* in a graph $G = (V, E)$ is a partition $(B_1, B_2, \dots, B_k)$ of V into k sets such that $\lfloor \frac{n}{k} \rfloor \leq |B_\ell| \leq \lceil \frac{n}{k} \rceil$ for all $\ell \in [k]$, where n denotes the number of vertices of G. The *width* of a k-section $(B_1, \dots, B_k)$ in a graph $G = (V, E)$ is the number of edges $\{x, y\} \in E$ such that x and y belong to different sets B_ℓ and $B_{\ell'}$ and is denoted by $e_G(B_1, \dots, B_k)$. A k-section of minimum width in a graph G is called a *minimum k-section* and its width is denoted by $\mathrm{MinSec}_k(G)$. So, $\mathrm{MinBis}(G) = \mathrm{MinSec}_2(G)$ for all graphs G. A path P on n vertices satisfies $\mathrm{MinSec}_k(P) = k - 1$ for all integers $2 \leq k \leq n$ and it is easy to verify that $\mathrm{MinSec}_k(K_n)$ increases as k increases. Intuitively, one would expect that, when considering one fixed graph G, then $\mathrm{MinSec}_k(G)$ increases as k increases, as it seems necessary to cut more edges when partitioning the vertex set into more classes. This is wrong, even when G is a perfect binary tree, as an example by Feldmann and Foschini [FF15] shows. Figure 1.2 presents a similar example: Let T be a perfect ternary tree of height h. If k is an even constant, then any k-section in T cuts at least $\mathrm{MinBis}(T)$ edges, i. e., at least $h - \log_3(h)$ edges as mentioned above. If k is a constant that is a power of 3, then T allows a k-section that cuts only edges in the upper $\log_3(k)$ levels of T, i. e., that cuts only a constant number of edges.

Minimum k-Section denotes the problem to determine $\mathrm{MinSec}_k(G)$ when given the graph G and an integer $k \geq 2$ as input. As mentioned above, the Minimum Bisection Problem is NP-hard and, hence, the Minimum k-Section Problem is NP-hard as well. The following simple reduction shows that, for arbitrary, fixed $k \geq 3$, the Minimum k-Section Problem is NP-hard as well. Let G be a graph on n vertices and assume that n is even. Let G' be the graph obtained from G by adding two copies of K_{n^2} and $k - 2$ copies of $K_{n^2 + \frac{1}{2}n}$. Then, a k-section in G' can be constructed from a bisection (B, W) in G by adding the copies of K_{n^2} to the sets B and W, respectively, and putting each copy of $K_{n^2+\frac{1}{2}n}$ into a set by itself. Thus, $\mathrm{MinSec}_k(G') \leq \mathrm{MinBis}(G)$. Furthermore, any cut $(\hat{B}, \hat{W})$ with $\hat{B} \neq \emptyset$ and $\hat{W} \neq \emptyset$ in a clique on at least n^2 vertices cuts at least $n^2 - 1$ edges. As the subgraph $G \subseteq G'$ has at most $\binom{n}{2} = \frac{1}{2}n(n-1)$ edges, every minimum k-section $(B_1, \dots, B_k)$ in G' cuts only edges in G and induces a bisection in G. Consequently, $\mathrm{MinSec}_k(G') = \mathrm{MinBis}(G)$ and solving the Minimum k-Section Problem for a fixed $k \geq 3$ is as hard as solving the Minimum Bisection Problem.

Returning to the case when k is part of the input, Andreev and Räcke [AR06] showed that, for general graphs, it is NP-hard to approximate the width of a minimum k-section within a finite factor. For trees, the dynamic programming algorithm in Theorem 4.3 in [Jan+05] to compute a minimum bisection can be adapted to compute a minimum k-section in a tree T such that the running time is polynomial in n

but not in k, where n denotes the number of vertices of T. Feldmann and Foschini [FF15] showed that the Minimum k-Section Problem remains APX-hard when restricted to trees with bounded degree. Additionally, they showed that it is NP-hard to approximate $\operatorname{MinSec}_k(T)$ within a factor of n^c for any $c < 1$ for trees T on n vertices with constant diameter. Note that, for a tree T on $n \geq 2$ vertices, an approximation of $\operatorname{MinSec}_k(T)$ within a factor of n is trivial as any k-section $(B_1, \ldots, B_k)$ in T satisfies $1 \leq e_T(B_1, \ldots, B_k) \leq n - 1$.

The spectral lower bound for the minimum bisection width can be generalized for the minimum k-section width in the following way. Let G be an arbitrary graph on n vertices and denote by $\lambda_1, \ldots, \lambda_k$ the k smallest eigenvalues of the Laplacian of G. Then, $\operatorname{MinSec}_k(G) \geq \frac{n}{2k} \sum_{i=1}^{k} \lambda_i(G)$, see [ELM03] where this bound is also improved to obtain a tight bound for certain graph classes.

1.1.3 Related Problems and Applications

One problem closely related to the Minimum Bisection Problem is the Maximum Bisection Problem, whose goal is to find a bisection (B, W) in a given graph G such that $e_G(B, W)$ is maximized. This problem is also NP-hard, as one can consider the complement of the graph to obtain an instance of the Maximum Bisection Problem from an instance of the Minimum Bisection Problem and vice versa. Nevertheless, in terms of approximation, the Minimum Bisection Problem and the Maximum Bisection Problem seem to behave very differently, as there is a 0.699-approximation for the Maximum Bisection Problem [Ye01], whereas no constant-factor approximation is known for the Minimum Bisection Problem. To understand this different behavior better, recall the approach of randomly choosing the set B for a bisection (B, W) in a given graph $G = (V, E)$ with $n := |V|$ even, which produces a bisection of expected width $\frac{1}{2}|E|\frac{n}{n-1} \approx \frac{1}{2}|E|$. This approach neither aims to minimize nor to maximize the number of edges between the sets B and W. On the one hand, a bisection (B, W) in G with $e_G(B, W) = \frac{1}{2}|E|$ can cut arbitrarily more edges than a minimum bisection in G or, more formally, there is no $\alpha \in \mathbb{R}$ with $e_G(B, W) \leq \alpha \operatorname{MinBis}(G)$ for all such graphs G and all such bisections (B, W), as $\operatorname{MinBis}(G) = 0$ is possible. On the other hand, a bisection (B, W) in G with $e_G(B, W) = \frac{1}{2}|E|$ cuts at least half of the number of edges cut by a maximum bisection in G since the width of any bisection in G is obviously bounded above by $|E|$. So, an algorithm computing a bisection (B, W) in a given graph G such that (B, W) satisfies $e_G(B, W) \geq \frac{1}{2}|E|$ is a $\frac{1}{2}$-approximation for the Maximum Bisection Problem.

Another closely related problem is the Minimum Cut Problem, which is, roughly speaking, obtained from the Minimum Bisection Problem by dropping the requirement that the sets B and W of a bisection (B, W) need to have the same size. More formally, the aim of the Minimum Cut Problem is to partition the vertex set of a given graph G into two non-empty sets B and W such that $e_G(B, W)$ is minimized. Surprisingly, there are efficient algorithms known for this problem [HO92] but the Maximum Cut Problem, that is similarly obtained from the Maximum Bisection Problem by removing the size constraint, is NP-hard, see Problem ND16 in [GJ79].

Feldmann and Foschini [FF15] study a version of the Minimum k-Section Problem, that asks to partition the vertex set of a given graph G on n vertices into k sets $B_1, \ldots, B_k$ such that $|B_\ell| \leq (1+\varepsilon) \left\lceil \frac{n}{k} \right\rceil$ for all $\ell \in [k]$, where k and $\varepsilon > 0$ are part of the input. They show that, for fixed $\varepsilon > 0$ and general graphs G on n vertices, a solution cutting at most $\mathcal{O}(\log n)$ times as many edges as a minimum k-section in G can be computed in polynomial time. For the case $\varepsilon = 1$, Krauthgamer et al. [KNS09] present a polynomial-time algorithm that, for graphs G on n vertices, computes a solution cutting at most $\mathcal{O}(\sqrt{\log n \log k})$ times as many edges as a minimum k-section in G.

The Minimum Bisection Problem and the Minimum k-Section Problem have many applications, some of which are demonstrated by the following examples. In the area of parallel computing, a fixed number of processors is available and one aims to distribute the computational tasks evenly to the processors while minimizing the communication cost between the processors. Such scenarios are typical for finite element simulations and can be modeled in the following way. Let G be a graph, where each vertex represents one computational task and where two vertices are adjacent if and only if the corresponding tasks depend on each other, i. e., one task can only be executed once the other task is finished. Then, the aim is to find a k-section in G of minimum width, where k is the number of processors. See also [Fel13] and the reference therein for a concrete example.

Divide-and-conquer approaches are a basic technique for developing algorithms. The main idea is to split the problem into two subproblems of roughly half the size of the original problem that are as independent as possible. Then, the subproblems are solved recursively, and finally the solutions for both subproblems are combined to one solution for the original problem. As a toy example, consider a graph G in which we want to count the triangles. Using a bisection (B, W) in G, one can split the problem into the two subproblems $G[B]$ and $G[W]$, and solve these recursively. To compute the number of triangles in G, one adds up the number of triangles in $G[B]$ and $G[W]$, and counts the triangles in G with vertices in both B and W. Now, the smaller the number of edges between B and W in G, the less work has to be done to count the triangles with vertices in B and W. Hence, it is desirable to use a minimum bisection to split G. Since the number of triangles in a graph can be computed in polynomial time, it does not make sense to solve this problem with a divide-and-conquer approach, which relies on computing a bisection of small width in each step. In Chapter 5.1 in [Shm97], a divide-and-conquer approach for the NP-hard problem to compute a minimum cut linear arrangement is presented and the dependence of the approximation ratio of the corresponding algorithm on the approximation ratio of the algorithm used for bisecting the graph is determined.

In VLSI Design, thousands of transistors are combined into a single chip in order to create an integrated circuit. To find a layout of the chip, usually, the area of the chip is divided into smaller parts and the circuit is divided accordingly while the interaction between the parts of the circuit needs to be minimized. This is important for the reliability of the chip and also to minimize the propagation delay. Partitioning the circuit can be modeled as follows. For each component of the circuit, there is a vertex and two vertices are adjacent whenever the circuit contains a connection between the corresponding components. Then, minimizing the interactions between the parts of the circuit means to find a k-section in the corresponding graph. See [BL84] and [PS96] for more information.

While the above examples can be modeled with unweighted graphs, there are many real world applications that require vertex weights or edge weights or even both. One such example is the consolidation of farmland as studied in [BBG11]. In typical farming areas, farmers cultivate a large number of small-sized lots, which are scattered over an extended area. This is disadvantageous as large machinery cannot be utilized and much time is needed for going back and forth between the lots. Consider a graph that has a vertex for each lot and two vertices are adjacent if the corresponding lots have a common border. In order to redistribute the lots between farmers, it obviously makes sense to use vertex weights for modeling the sizes or values of the lots. Furthermore, edge weights can be used to model the length of a common border of the corresponding lots, disregarding parts of the border that are natural as streets or rivers. Let k be the number of farmers. Then, assuming that every farmer owns the same total amount of farmland, the goal is to compute a k-section of small width. Observe that the edge weights used in [BBG11] are different as a clustering approach is used.

Last but not least, observe that, in the presented applications, the size constraints can be relaxed by considering the problem of partitioning the vertex set of a given graph on n vertices into k sets $B_1, B_2, \dots, B_k$ with $|B_\ell| \leq (1+\varepsilon)\left\lceil \frac{n}{k} \right\rceil$ for all $\ell \in [k]$. The benefit of this relaxation is that there are polynomial-time algorithms that compute solutions whose width is not too far from a minimum k-section as discussed above.

1.2 Overview of Results

This thesis focuses on bounded-degree graphs, but usually the dependence on the maximum degree is stated explicitly by using $\Delta(G)$ to denote the *maximum degree* of a graph G. Hence, the results hold for arbitrary graphs, but most results are only interesting when the maximum degree is low. The MINIMUM BISECTION PROBLEM is still interesting when only bounded-degree graphs are considered, since it remains NP-hard when restricted to graphs with maximum degree 3 as mentioned in Section 1.1.1. Furthermore, Berman and Karpinski [BK02] showed that the MINIMUM BISECTION PROBLEM restricted to 3-regular graphs is as hard to approximate as its general version.

1.2.1 Structural Results for Trees and Tree-Like Graphs with Large Minimum Bisection Width

One aim of this thesis is to investigate the structure of graphs with large minimum bisection width. Here, bounded-degree trees and bounded-degree tree-like graphs are studied, whereas Section 1.2.2 focuses on planar graphs. Let T be a tree on n vertices. As mentioned in Section 1.1.1, T allows a bisection of width at most $\Delta(T)\log_2(n)$. A family $\mathcal{T}$ of trees T is called a family of bounded-degree trees with *large minimum bisection width* if there are two constants $\Delta_0 \in \mathbb{N}$ and $c > 0$ such that $\Delta(T) \leq \Delta_0$ and $\operatorname{MinBis}(T) \geq c\log_2(|V(T)|)$ for every $T \in \mathcal{T}$. For better readability, we abuse notation in the following and use the term "large minimum bisection width" also for single trees. The following inequality is established in order to analyze the structure of bounded-degree trees with large minimum bisection width. There, $\operatorname{diam}(T)$ denotes the *diameter* of T, which is defined as the number of edges in a longest path in the tree T.

Theorem 1.1.
Every tree T on n vertices satisfies

$$\operatorname{MinBis}(T) \;\leq\; \frac{8n\Delta(T)}{\operatorname{diam}(T)}.$$

At first sight, this theorem might not look interesting compared to the bound $\operatorname{MinBis}(T) \leq \Delta(T)\log_2(n)$ as the bound in Theorem 1.1 contains a factor n. However, if T is a bounded-degree tree with linear diameter, the bound in Theorem 1.1 implies that T allows a bisection of constant width. So, a bounded-degree tree with large minimum bisection width cannot have linear diameter. More precisely, fix $\Delta_0 \in \mathbb{N}$ and $c > 0$, and consider an arbitrary tree T on n vertices with maximum degree at most Δ_0 and $\operatorname{MinBis}(T) \geq c\log_2(n)$. Then, Theorem 1.1 implies that $c\log_2(n) \leq \frac{8n\Delta_0}{\operatorname{diam}(T)}$ or, equivalently, $\operatorname{diam}(T) \leq \frac{8n\Delta_0}{c\log_2(n)}$ and the next corollary follows.

Corollary 1.2.
For every $\Delta_0 \in \mathbb{N}$ and every $c > 0$, there is an $\alpha > 0$ such that the following holds:
If T is a tree on n vertices with maximum degree at most Δ_0 and $\operatorname{MinBis}(T) \geq c\log_2(n)$, then T does not contain a path of length $\frac{\alpha n}{\log_2(n)}$ or greater.

The bound in Theorem 1.1 can be generalized to arbitrary graphs with a given tree decomposition $(T, \mathcal{X})$. Instead of considering a longest path in the underlying graph G, a parameter $r(T, \mathcal{X})$, that roughly measures how close the tree decomposition $(T, \mathcal{X})$ is to a path decomposition, is defined. For example, every path P on n vertices satisfies $\frac{1}{n}\operatorname{diam}(P) = \frac{n-1}{n} \approx 1$ and allows a bisection of width 1. When the diameter of a tree decreases, it looks less like a path. Similarly, consider a graph G on n vertices and a path decomposition $(P, \mathcal{X})$ of G of width $t-1$. One can show that G allows a bisection of width at most $t\Delta(G)$ by walking along the path P until we have seen $\lfloor \frac{1}{2}n \rfloor$ vertices of G in the clusters and then bisecting G there, see also Theorem 7.11 in [Sch13]. Therefore, we will define $r(T, \mathcal{X})$ such that $r(T, \mathcal{X}) = 1$ for path decompositions $(T, \mathcal{X})$ and such that $r(T, \mathcal{X})$ decreases when $(T, \mathcal{X})$ looks less like a path decomposition. Let $G = (V, E)$ be a graph on n vertices and consider a tree decomposition $(T, \mathcal{X})$ of G with $\mathcal{X} = (X^i)_{i \in V(T)}$. The *relative weight* of a path $P \subseteq T$ and the *relative weight of a heaviest path* in $(T, \mathcal{X})$ are defined as

$$w^*_{\mathcal{X}}(P) \;:=\; \frac{1}{n}\left|\bigcup_{i \in V(P)} X^i\right| \qquad \text{and} \qquad r(T, \mathcal{X}) \;:=\; \max_{P \text{ path in } T} w^*_{\mathcal{X}}(P),$$

respectively. Observe that every tree decomposition $(T, \mathcal{X})$ satisfies $\frac{1}{n} \le r(T, \mathcal{X}) \le 1$, where n denotes the number of vertices of the underlying graph. One can show that every tree T' allows a tree decomposition $(T, \mathcal{X})$ with $r(T, \mathcal{X}) \ge \frac{\operatorname{diam}(T')}{n'}$, where n' denotes the number of vertices of T'. So, the parameter $r(T, \mathcal{X})$ for a tree decomposition $(T, \mathcal{X})$ corresponds to the fraction $\frac{\operatorname{diam}(T')}{|V(T')|}$ for trees T'.

Theorem 1.3.
Every graph G that allows a tree decomposition $(T, \mathcal{X})$ of width $t-1$ satisfies

$$\operatorname{MinBis}(G) \;\le\; \frac{8t\Delta(G)}{r(T, \mathcal{X})}.$$

Fix $\Delta_0 \in \mathbb{N}$ and consider a graph G on n vertices with maximum degree at most Δ_0 and $\operatorname{tw}(G) \ge 1$. The bound from Section 1.1.1 implies that $\operatorname{MinBis}(G) = \mathcal{O}(\operatorname{tw}(G) \cdot \log n)$ and we say that G has *large minimum bisection width* if $\operatorname{MinBis}(G) = \Omega(\operatorname{tw}(G) \cdot \log n)$. Assume that G has large minimum bisection width, i. e., there is a constant $c > 0$ such that $\operatorname{MinBis}(G) \ge c\operatorname{tw}(G) \cdot \log_2(n)$. Then, Theorem 1.3 implies that, for every tree decomposition $(T, \mathcal{X})$ of G of width $t-1$,

$$c \cdot \operatorname{tw}(G) \cdot \log_2(n) \;\le\; \frac{8t\Delta_0}{r(T, \mathcal{X})},$$

which is equivalent to

$$r(T, \mathcal{X}) \;\le\; \frac{8t\Delta_0}{c \cdot \operatorname{tw}(G) \cdot \log_2(n)}.$$

Hence, every tree decomposition $(T, \mathcal{X})$ of G of width $\mathcal{O}(\operatorname{tw}(G))$ must satisfy $r(T, \mathcal{X}) = \mathcal{O}\left(\frac{1}{\log n}\right)$. In particular, this implies that every tree decomposition $(T, \mathcal{X})$ of G of minimum width satisfies $r(T, \mathcal{X}) = \mathcal{O}\left(\frac{1}{\log n}\right)$ and is far from being a path decomposition of G.

Before continuing with planar graphs, observe that the bounded maximum degree is necessary due to the following example. Let T be the tree obtained from a star on $\frac{3}{4}n + 1$ vertices by attaching a path on $\frac{1}{4}n - 1$ vertices to its center vertex. Then $\operatorname{MinBis}(T) \ge \frac{1}{4}n$ as each set of the bisection must contain at least $\frac{1}{4}n$ leaves of the star but its center vertex is only in one of the sets of the bisection. So T is a tree with large minimum bisection width, except for having bounded degree, but T contains a path of linear length.

1.2.2 Structural Results for Planar Graphs with Large Minimum Bisection Width

Turning our attention to planar graphs, observe first that Theorem 1.1 does not hold for planar graphs. Indeed, consider the planar graph G obtained from a square grid on $\frac{3}{4}n$ vertices and a path on $\frac{1}{4}n$ vertices attached to one of the corner vertices of the grid. Then $\operatorname{diam}(G) \geq \frac{1}{4}n$ and $\Delta(G) = 4$. So, if Theorem 1.1 was true for planar graphs, then there would be a bisection of constant width in G. However, for every bisection (B, W) in G, the sets B and W each contain between $\frac{1}{4}n$ and $\frac{1}{2}n$ vertices of the grid and Theorem 6 in [LT79] implies that $\Omega(\sqrt{n})$ edges of the grid are cut by the bisection (B, W). So, how large can the minimum bisection width of a bounded-degree planar graph be? Every planar graph G on n vertices satisfies $\operatorname{MinBis}(G) \leq 16 \cdot \Delta(G) \cdot \sqrt{n}$ as mentioned in Section 1.1.1, and the previous example shows that this bound is tight up to a constant factor. We say that a family $\mathcal{G}$ of planar graphs has *large minimum bisection width* if there are two constants $\Delta_0 \in \mathbb{N}$ and $c > 0$ such that $\Delta(G) \leq \Delta_0(G)$ and $\operatorname{MinBis}(G) \geq c\sqrt{|V(G)|}$ for every graph $G \in \mathcal{G}$. Similarly to trees with large minimum bisection width, this term is also used for single graphs for better readability. The following theorem for bounded-degree planar graphs with large minimum bisection width is derived here.

Theorem 1.4.
For every $\Delta_0 \in \mathbb{N}$ and every $c > 0$, there is a $\gamma > 0$ such that the following holds:
If G is a planar graph on n vertices with maximum degree at most Δ_0 and $\operatorname{MinBis}(G) \geq c\sqrt{n}$, then the tree-width of G is at least $\gamma\sqrt{n} - 1$.

So, roughly speaking, every bounded-degree planar graph that has large minimum bisection width is far from being tree-like. The fact that every planar graph G contains a $k \times k$ grid with $k = \left\lfloor \frac{1}{6}(\operatorname{tw}(G) + 4) \right\rfloor$ as a minor, which can be found in the famous work by Robertson, Seymour, and Thomas on graph minors in Theorem 6.2 in [RST94], implies the following corollary.

Corollary 1.5.
For every $\Delta_0 \in \mathbb{N}$ and every $c > 0$, there is a $\gamma' > 0$ such that the following holds:
If G is a planar graph on n vertices with maximum degree at most Δ_0 and $\operatorname{MinBis}(G) \geq c\sqrt{n}$, then G contains a $k \times k$ grid with $k = \lfloor \gamma' \sqrt{n} \rfloor$ as a minor.

As in Section 1.2.1, the star on n vertices shows that the bounded maximum degree is necessary in Theorem 1.4 and Corollary 1.5. Every planar graph on n vertices has tree-width $\mathcal{O}(\sqrt{n})$, see Proposition 2.13 in the next chapter, and the largest integer k for which a planar graph on n vertices can contain a $k \times k$ grid as a minor satisfies $k = \mathcal{O}(\sqrt{n})$. So, we say that a planar graph on n vertices has *large tree-width* if it has tree-width $\Omega(\sqrt{n})$, and we say that a planar graph on n vertices *contains a large grid as minor* if it contains a $k \times k$ grid with $k = \Omega(\sqrt{n})$ as a minor. Again, both terms should only be used for families of graphs, but are used for single graphs for better readability. To summarize Theorem 1.4 and Corollary 1.5, for bounded-degree planar graphs, having large minimum bisection width implies having large tree-width, and, for planar graphs, the latter is equivalent to containing a large grid as a minor.

One of the remaining questions is whether, for bounded-degree planar graphs, having large tree-width or, equivalently, containing a large grid as a minor implies having large minimum bisection width. This implication is not true as the graph consisting of two disjoint $k \times k$ grids has minimum bisection width zero and contains a large grid as a minor. The following question emerges: What additional properties force bounded-degree planar graphs that contain a large grid as a minor to have large minimum bisection width? To answer this question, which does not seem to have a simple, straightforward answer, the concept of grid-homogeneous graphs is introduced. The idea behind it is that a grid-homogeneous graph

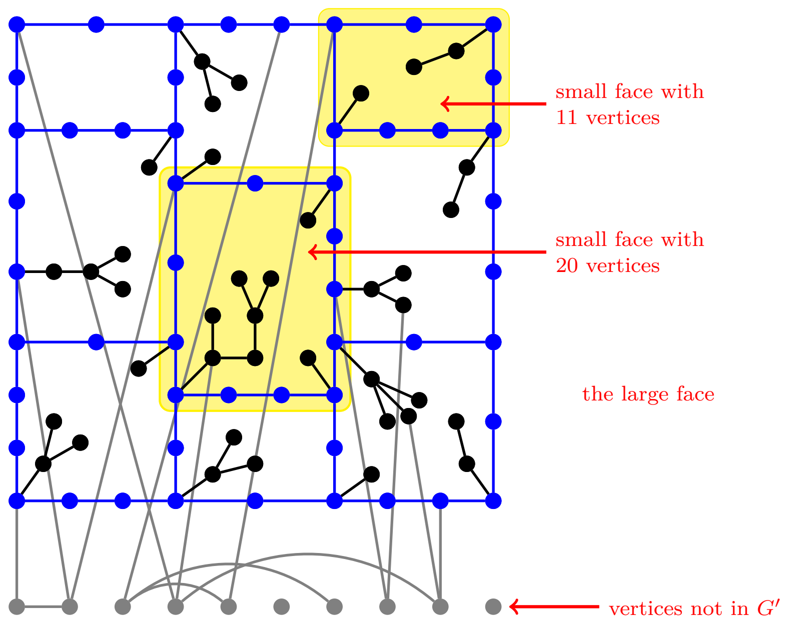


Figure 1.3: Example of a grid-homogeneous graph G. Subgraphs H and G' with the properties required by Definition 1.6 are highlighted. The subgraph H is colored blue, vertices and edges in the subgraph G' that do not belong to H are colored black. Vertices and edges that are in G but not in G' are colored gray. Even though Definition 1.6 and Theorem 1.7 require $k \geq 5$, we chose $k = 4$ to keep the example small.

should contain a connected subgraph on almost all its vertices and a large grid minor that is spread homogeneously through the subgraph. Thus, every bisection in a grid-homogeneous graph needs to cut off many vertices from a graph that behaves similarly to a grid.

Definition 1.6 ((γ, k, ℓ)-grid-homogeneous).
Let $k, \ell \in \mathbb{N}$ with $k \geq 5$ and $0 \leq \gamma < 1$. A graph $G = (V, E)$ is called (γ, k, ℓ)-*grid-homogeneous* if it contains a connected planar graph $G' = (V', E') \subseteq G$ with $|V'| \geq (1 - \gamma)|V|$ and a graph $H = (V_H, E_H) \subseteq G'$ as subgraphs such that G' has an embedding in the plane with the following properties:

- The graph H is a minimal graph containing a $k \times k$ grid as a minor.
- For every small face f of the induced embedding of H, at most ℓ vertices from V' are embedded in the face f including the vertices on its boundary.
- No vertex from $V' \setminus V_H$ is embedded in the large face f of the induced embedding of H.

Figure 1.3 shows an example of a grid-homogeneous graph and gives an intuition for the terms small and large face, which are defined formally in Section 3.2.2. There, we will also argue that every property required for the graph G' in Definition 1.6 is necessary in order to prove a lower bound for the minimum bisection width in grid-homogeneous graphs as in the next theorem.

Theorem 1.7.
For every $k, \ell \in \mathbb{N}$ with $k \geq 5$ and every $0 \leq \gamma < \frac{1}{2}$, every (γ, k, ℓ)-grid-homogeneous graph $G = (V, E)$ with $|V|$ even satisfies

$$\mathrm{MinBis}(G) \;\geq\; \left(\tfrac{1}{2} - \gamma\right) \frac{1}{4\ell} k.$$

Fix an arbitrary γ with $0 \leq \gamma < \frac{1}{2}$, an integer $\ell \in \mathbb{N}$, an integer $\Delta_0 \in \mathbb{N}$, and a $c > 0$. Consider a family $\mathcal{G}$ of planar graphs such that each graph $G \in \mathcal{G}$ on n vertices is (γ, k, ℓ)-grid-homogeneous with $k \geq c\sqrt{n}$

and satisfies $\Delta(G) \leq \Delta_0$. Then, Theorem 1.7 implies that there is a constant $c_1 > 0$ such that every graph $G \in \mathcal{G}$ on n vertices satisfies $\mathrm{MinBis}(G) \geq c_1\sqrt{n}$, i. e., the family $\mathcal{G}$ has large minimum bisection width. On the other hand, Theorem 1.10, which is presented further ahead, says that there is a constant c_2 such that every graph $G \in \mathcal{G}$ on n vertices satisfies $\mathrm{MinBis}(G) \leq c_2\sqrt{n}$ and that a bisection of width within this bound can be computed in linear time. Therefore, the algorithm contained in Theorem 1.10 is a constant-factor approximation for the Minimum Bisection Problem when restricted to the class $\mathcal{G}$.

However, the algorithmic use of Theorem 1.7 is limited as some questions related to asking whether a graph G is (γ, k, ℓ)-grid-homogeneous for certain parameters γ, k, and ℓ are NP-complete. In particular, it is shown in Section 3.3 that, for every $\ell \geq 6$, it is NP-complete to decide, when given a planar graph G' and a plane subgraph H that is a minimal graph containing a $k \times k$ grid as a minor, whether G' allows an embedding in the plane that is an extension of the embedding of H, such that no vertex from $V(G') \setminus V(H)$ is embedded in the large face of H and, for each small face f of H, there are at most ℓ vertices from $V(G')$ embedded in f including the vertices on the boundary of f. Furthermore, in Section 3.3, it is shown that finding the smallest integer ℓ such that G' allows such an embedding is NP-hard to approximate within any constant $< \frac{3}{2}$. When requiring additionally that the considered graph G' does not contain more edges than necessary to be a connected graph containing H as a subgraph, an algorithm approximating the smallest such integer ℓ within a factor of 2 can be obtained by restating the problem as a scheduling problem for unrelated parallel machines. Moreover, it is shown in Section 3.3 that it is NP-complete to decide, when given a planar graph G' that is uniquely embeddable and two integers ℓ and k, whether G' contains a subgraph H that is a minimal graph containing a $k \times k$ grid as a minor such that no vertex from $V(G') \setminus V(H)$ is embedded in the large face of H and, for every small face f of H, at most ℓ vertices from $V(G')$ are embedded in f including the vertices on the boundary of f.

1.2.3 Algorithmic Results for Bisections

The proofs of Theorem 1.1 and Theorem 1.3 are mostly constructive. So it is natural to ask whether there are polynomial-time algorithms for computing bisections of width within the bounds mentioned there. Moreover, we would like to see whether the bounds on the width of the bisections can be improved. Both questions are answered positively by the next theorems.

Theorem 1.8 (improved and algorithmic version of Theorem 1.1).
For every tree T on n vertices, a bisection (B, W) in T satisfying

$$e_T(B,W) \;\leq\; \frac{\Delta(T)}{2}\left(\left(\log_2\left(\frac{n}{\mathrm{diam}(T)}\right)\right)^2 + 7\log_2\left(\frac{n}{\mathrm{diam}(T)}\right) + 6\right)$$

can be computed in $\mathcal{O}(n)$ time.

Recall that when generalizing Theorem 1.1 to tree-like graphs, a tree decomposition was used. Since it is NP-hard to determine the tree-width of a graph [ACP87], the algorithm in the following theorem receives a tree decomposition as input. For a tree decomposition $(T, \mathcal{X})$ with $\mathcal{X} = (X^i)_{i \in V(T)}$, the *size* of $(T, \mathcal{X})$ is defined as $\|(T, \mathcal{X})\| := |V(T)| + \sum_{i \in V(T)} |X^i|$, which is proportional to the encoding length of $(T, \mathcal{X})$. Hence, a linear-time algorithm that receives a tree decomposition $(T, \mathcal{X})$ as input runs in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time.

Theorem 1.9 (improved and algorithmic version of Theorem 1.3).
For every graph G and for every tree decomposition $(T, \mathcal{X})$ of G of width at most $t-1$, a bisection (B, W) in G with

$$e_G(B,W) \;\leq\; \frac{t\Delta(G)}{2}\left(\left(\log_2\left(\frac{1}{r(T,\mathcal{X})}\right)\right)^2 + 9\log_2\left(\frac{1}{r(T,\mathcal{X})}\right) + 8\right)$$

can be computed in $\mathcal{O}(\|(T,\mathcal{X})\|)$ time when the tree decomposition $(T, \mathcal{X})$ is provided as input.

Observe that neither the algorithm contained in Theorem 1.8 nor the algorithm contained in Theorem 1.9 computes a minimum bisection, but a bisection of small width is computed for certain trees and tree-like graphs, respectively. Theorem 1.8 says that bounded-degree trees with linear diameter allow a bisection of constant width and Theorem 1.9 implies that every bounded-degree graph G that allows a tree decomposition $(T, \mathcal{X})$ of constant width and with $r(T, \mathcal{X}) = \Omega(1)$ satisfies $\text{MinBis}(G) = \mathcal{O}(1)$. Moreover, both algorithms run in linear time and the algorithm contained in Theorem 4.3 in [Jan+05] takes $\mathcal{O}(2^t n^3)$ time to compute a minimum bisection in a graph on n vertices when given a tree decomposition of width t.

The proof of Theorem 1.4 also has a constructive part. More precisely, Section 3.1.4 derives an upper bound for the width of a minimum bisection in bounded-degree planar graphs that depends on the tree-width of the graph. The main idea is a general method for constructing a bisection via separators. In a graph $G = (V, E)$ on n vertices, a set $S \subseteq V$ is a *separator* if every component of $G - S$ contains at most $\frac{1}{2}n$ vertices. To construct a bisection in a graph, the graph can be split into smaller parts by removing a separator and then putting components greedily into an initially empty black set, until one component G' is reached such that the black set and G' together contain more than half of the vertices of the initial graph. All remaining components, except G', fit into the white set. Then, G' is split into smaller parts by removing a separator and again components are greedily put into the black set and so on. As the size of the considered graph shrinks in each round, at one point, each vertex of the initial graph is either in the black set, the white set, or has been removed. Additionally, the black and the white set each contain at most half of the vertices of the initial graph. Then, it is possible to distribute all removed vertices to the black and the white set such that a bisection in the initial graph is obtained. One can show that its width is at most the number of removed vertices times the maximum degree. Applying this approach with planar separators, see [LT79], and separators constructed from tree decompositions yields the following theorems, where a tree decomposition $(T, \mathcal{X})$ with $\mathcal{X} = (X^i)_{i \in V(T)}$ is called *nonredundant* if $X^i \not\subseteq X^j$ and $X^j \not\subseteq X^i$ for every edge $\{i, j\} \in E(T)$. Furthermore, define $\|G\| := |V(G)| + |E(G)|$.

Theorem 1.10.
There is a constant c_σ such that, for every planar graph G on n vertices, a bisection (B, W) in G satisfying

$$e_G(B,W) \;\leq\; c_\sigma \Delta(G) \cdot \sqrt{n}$$

can be computed in $\mathcal{O}(n)$ time.

Theorem 1.11.
For every graph G on n vertices, and every tree decomposition $(T, \mathcal{X})$ of G of width $t-1$, a bisection (B, W) in G satisfying

$$e_G(B,W) \;\leq\; t\Delta(G) \cdot \log_2(n)$$

can be computed in $\mathcal{O}\left((\|G\| + \|(T,\mathcal{X})\|)\log_2(n)\right)$ time if the tree decomposition $(T, \mathcal{X})$ is provided as input. If the provided tree decomposition is nonredundant, the running time is $\mathcal{O}(nt)$.

Theorem 6.2 in [Jan+05] states a similar bound for the width of a minimum bisection in a planar graph as Theorem 1.10, but the running time in [Jan+05] is $\mathcal{O}(n \log n)$ for a graph on n vertices. Hence, Theorem 1.10 is an algorithmic improvement. Combining the separators used in the proofs of Theorem 1.10 and Theorem 1.11 gives a technical upper bound for the minimum bisection width in planar graphs G that depends on the tree-width of G and is the key idea to prove Theorem 1.4.

1.2.4 Approximate Cuts in Tree-Like Graphs

When proving Theorem 1.1 and Theorem 1.3, the following lemmas that relax the size constraints of a bisection are useful. First of all, not only bisections but m-cuts, which are partitions (B, W) of the vertex set of the considered graph with $|B| = m$, are studied. To relax the size constraint, consider a constant $0 < c < 1$, let G be a graph on n vertices, and fix an integer $m \in [n]$. A partition (B, W) of the vertex set of G is a *c-approximate m-cut* in G if $cm \leq |B| \leq m$. The following lemmas state that trees and tree-like graphs allow approximate cuts of small width and that they can be computed in linear time. In both lemmas, the hidden constant in the running time does not depend on c.

Lemma 1.12 (Approximate Cut in Trees).
For every $0 < c < 1$, for every tree T on n vertices, and for every $m \in [n]$, a c-approximate m-cut (B, W) in T with

$$e_T(B, W) \;\leq\; \Delta(T) \left\lceil \log_2 \left(\frac{1}{1-c} \right) \right\rceil$$

can be computed in $\mathcal{O}(n)$ time.

Lemma 1.13 (Approximate Cut in Tree-Like Graphs).
For every $0 < c < 1$, for every graph G on n vertices, and for every $m \in [n]$, the following holds: For every tree decomposition $(T, \mathcal{X})$ of G of width at most $t-1$, a c-approximate m-cut (B, W) in G with

$$e_G(B, W) \;\leq\; t\Delta(G) \left\lceil \log_2 \left(\frac{1}{1-c} \right) \right\rceil$$

can be computed in $\mathcal{O}\left(\|(T, \mathcal{X})\|\right)$ time, when $(T, \mathcal{X})$ is provided as input.

Fix a constant $0 < c < 1$ and consider a bounded-degree tree T on n vertices. Then, Lemma 1.12 implies that, for every $m \in [n]$, the tree T allows a c-approximate m-cut of constant width. Observe that for Theorem 1.8 to imply that T allows a bisection of constant width, it is necessary to require that T has linear diameter. This applies similarly to Theorem 1.9 and Lemma 1.13 when studying bounded-degree tree-like graphs.

Consider a c-approximate m-cut (B, W) in a graph G. The closer c is to 1, the closer the size of B is to m, but also the bounds on the width of c-approximate cuts in Lemma 1.12 and Lemma 1.13 increase as c approaches 1. When choosing c sufficiently close to 1, a c-approximate m-cut with $m = \left\lfloor \frac{1}{2}|V(G)| \right\rfloor$ is a bisection in G. This yields the next corollary, which is similar to Theorem 1.11. In particular the bound on the width of the bisection is the same, but the running time in the following corollary is asymptotically faster.

Corollary 1.14 (improved version of Theorem 1.11).
For every graph G on n vertices and for every tree decomposition $(T, \mathcal{X})$ of G of width at most $t-1$, a bisection (B, W) in G with

$$e_G(B, W) \;\leq\; t\Delta(G) \cdot \log_2(n)$$

can be computed in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time, when $(T, \mathcal{X})$ is provided as input.

Note that the algorithm in the previous corollary runs in linear time but none of the running times of the algorithm in Theorem 1.11 is linear in the input size for arbitrary values of t. For example, consider a tree decomposition $(T, \mathcal{X})$ of width $t-1$; if t is not constant and $(T, \mathcal{X})$ has only few clusters of size $\theta(t)$, then nt is asymptotically larger than $\|(T, \mathcal{X})\|$.

1.2.5 Minimum k-Section in Tree-Like Graphs

The algorithmic results for bisections presented in Section 1.2.3 can be generalized for computing k-sections. Generalizing Theorem 1.10 and Theorem 1.11 is straightforward by using the method described in [ST97]. The following theorem is a generalization of Theorem 1.1 for bisections in trees.

Theorem 1.15.
For every integer $k \geq 2$, for every tree T on n vertices, a k-section $(B_1, B_2, \ldots, B_k)$ in T with

$$e_T(B_1, B_2, \ldots, B_k) \;\leq\; (k-1)\Delta(T) \cdot \left(2 + \frac{16n}{\operatorname{diam}(T)}\right)$$

can be computed in $\mathcal{O}(kn)$ time.

Consider a graph G on n vertices and note that, if $k \geq n$, then every k-section in G cuts all edges of G and thus is a minimum k-section in G. Therefore, we can assume without loss of generality that the algorithm in the previous theorem is only applied for computing a k-section in a tree T with more than k vertices. In this case, the running time is $\mathcal{O}(n^2)$, which is polynomial in the size of the input.

Fix a constant $c > 0$ and an integer $\Delta_0 \geq 2$. Let $\alpha := \Delta_0 \left(2 + \frac{16}{c}\right)$. Then, Theorem 1.15 implies that, for every $k \geq 2$, every tree T on n vertices with $\Delta(T) \leq \Delta_0$ and $\operatorname{diam}(T) \geq cn$ satisfies $\operatorname{MinSec}_k(T) \;\leq\; \alpha(k-1)$ and that a k-section of width within this bound can be computed in polynomial time. As every k-section in T cuts at least $k-1$ edges, the algorithm contained in Theorem 1.15 computes a k-section $(B_1, \ldots, B_k)$ in T that satisfies $e_T(B_1, \ldots, B_k) \leq \alpha \operatorname{MinSec}_k(T)$ and is a constant-factor approximation for the Minimum k-Section Problem when restricted to trees with bounded degree and linear diameter. Recall that the Minimum k-Section Problem remains APX-hard when restricted to trees with bounded degree [FF15].

Although Theorem 1.1 and Theorem 1.15 look quite similar, it does not seem possible to directly apply Theorem 1.1 to yield a recursive construction of a k-section that satisfies the bound presented in Theorem 1.15. Indeed, it is known that, even when k is a power of 2, the natural approach to construct a k-section of a graph by recursively constructing bisections can yield a k-section far from a minimum k-section, even if a minimum bisection is used in each step [ST97]. Furthermore, in the setting that is considered here, i. e., bounded-degree trees with linear diameter, nothing is known about the diameter in the two subgraphs that are produced by an algorithm following the construction behind Theorem 1.1. So, after the first iteration, the diameter of one part of the bisection could be as low as $\mathcal{O}(\log n)$, and indeed such parts can be produced by such an algorithm as discussed in Section 6.1. This would then give a bound of $\Omega\left(\frac{n}{\log n}\right)$ for the bisection computed in the next round and also for the recursively computed 4-section, whereas Theorem 1.15 guarantees a 4-section of constant width for bounded-degree trees with linear diameter.

As Theorem 1.1 can be strengthened to Theorem 1.8, the bound on the width of the k-section in Theorem 1.15 can also be improved to

$$e_T(B_1, B_2, \ldots, B_k) \;\leq\; \tfrac{1}{2}(k-1)\;\Delta(T)\;\left(\left(\log_2\left(\frac{n}{\operatorname{diam}(T)}\right)\right)^2 + 9\log_2\left(\frac{n}{\operatorname{diam}(T)}\right) + 18\right).$$

Moreover, Theorem 1.8 can be generalized to arbitrary graphs by considering tree decompositions as done in Theorem 1.9, which can be used to prove the following theorem about k-sections in tree-like graphs.

Theorem 1.16.
For every integer $k \geq 2$, for every graph G and every tree decomposition $(T, \mathcal{X})$ of G of width at most $t-1$, a k-section $(B_1, B_2, \ldots, B_k)$ in G with

$$e_G(B_1, B_2, \ldots, B_k) \;\leq\; \tfrac{1}{2}(k-1)t\Delta(G)\left(\left(\log_2\left(\frac{1}{r(T,\mathcal{X})}\right)\right)^2 + 11\log_2\left(\frac{1}{r(T,\mathcal{X})}\right) + 24\right)$$

can be computed in $\mathcal{O}(k\|(T, \mathcal{X})\|)$ time, when the tree decomposition $(T, \mathcal{X})$ is provided as input.

1.2.6 Further Remarks

Most results from Sections 1.2.1-1.2.5 do not only hold for bisections but also for cuts (B, W) where the size of the set B is specified as input. When presenting the proof of such a result, it is stated in its general form.

Here, only bisections and k-sections in unweighted graphs are considered. In many applications, the vertices and the edges of the considered graph have weights. Any upper bound for the width of a minimum bisection in a graph without edge weights can be adjusted for graphs with edge weights by multiplying it with the maximum edge weight. For vertex-weights, the situation is more difficult. In a graph $G = (V, E)$ with vertex weights, deciding whether V allows a partition (B, W) such that the sum of the weights in B equals the sum of the weights in W is equivalent to solving the NP-complete Subset Sum Problem, see Problem MP9 in [GJ79]. In [Ham16], a bisection in a graph G with vertex weights is defined as a partition (B, W) of $V(G)$ such that $|g(B) - g(W)| \leq g_{\max}$, where $g(B)$ and $g(W)$ denote the sum of the weights in B and W, respectively, and $g_{\max}$ is the maximum vertex weight in G. Then, a version of Theorem 1.1 for trees with vertex weights and a corresponding algorithm is derived. Furthermore, in [Ham16], also Lemma 1.12 and Theorem 1.15 are generalized for trees with vertex weights.

1.3 Organization of the Thesis

After presenting some preliminaries in the next chapter, the proofs of the results summarized in Section 1.2 are presented in the following order.

Chapter 3 focuses on planar graphs. First, the method for constructing bisections in graphs via separators that was sketched in Section 1.2.3 is made precise and then used to derive Theorem 1.10 and Theorem 1.11. Furthermore, Theorem 1.4 is proved. Afterwards, Chapter 3 investigates bounded-degree planar graphs with large minimum bisection width. Grid-homogeneous graphs are introduced in detail and Theorem 1.7 is proved. Moreover, algorithmic aspects related to grid-homogeneous graphs as mentioned in Section 1.2.2 are discussed.

Chapter 4 studies approximate cuts in trees and tree-like graphs. In particular, Lemma 1.12 and Lemma 1.13 are proved here. Furthermore, Corollary 1.14 is derived in Chapter 4.

Chapter 5 concentrates on bisections in trees and tree-like graphs. First, the methods used throughout the chapter are introduced slowly by studying trees T on n vertices with $\operatorname{diam}(T) \geq \frac{1}{4}n$. Second, Theorem 1.1 is proved and an algorithm computing a bisection whose width is at most as large as the bound in Theorem 1.1 is described. Afterward, the analysis is tightened to yield Theorem 1.8. Third, the methods are generalized to tree-like graphs and Theorem 1.9 is proved. Observe that Theorem 1.3 does not need to be proved separately as Theorem 1.9 is stronger.

Chapter 6 focuses on k-sections. To begin with, it is shown that simple, recursive approaches using Theorem 1.1 to construct a k-section in a tree can yield a k-section of width much larger than promised by Theorem 1.15. Then, the proof of Theorem 1.15 is presented and the analysis is tightened as described in Section 1.2.5. Furthermore, the methods are generalized to tree-like graphs and Theorem 1.16 is derived.

Figure 1.4 visualizes connections between the results proved in this thesis. All results are joint work with Cristina G. Fernandes and Anusch Taraz. Parts of the results in Chapter 3 and parts of the results for trees in Chapter 5 have been published in the proceedings of EuroComb 2013 [FST13]. Some results on the Minimum k-Section Problem in trees from Chapter 6 have appeared in the proceedings of LAGOS 2015 [FST15b]. The extensions to tree-like graphs in Chapter 5 and Chapter 6 have been published in the proceedings of EuroComb 2015 [FST15a]. The journal version of the results from Chapter 4 and Chapter 5 has been submitted [FSTa]. Journal versions concerning the results of Chapter 3 and Chapter 6 are in preparation [FSTb; FSTc].

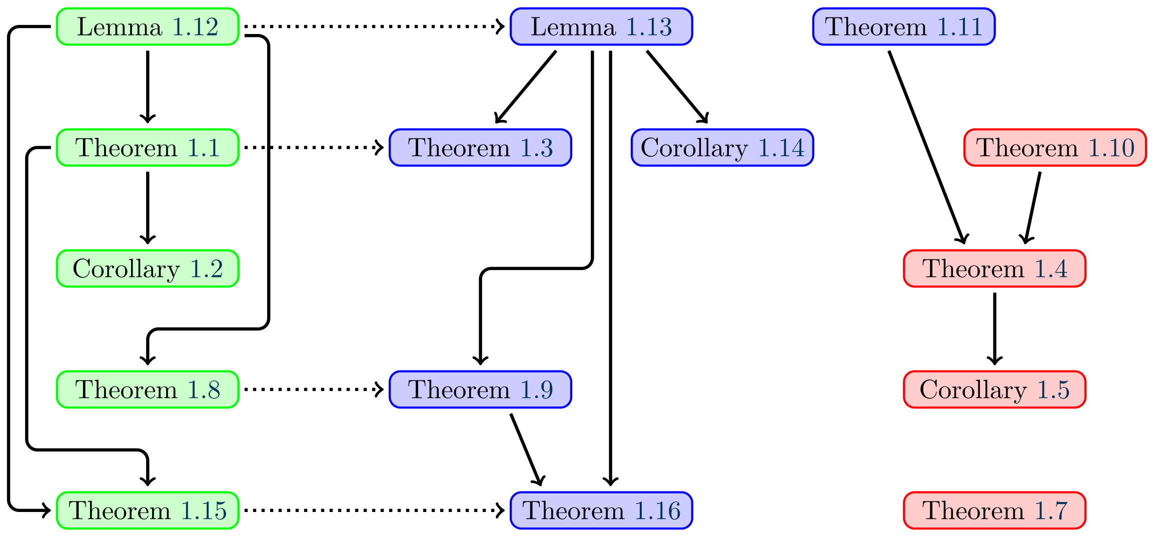


Figure 1.4: Visualization of the connections between the results proved in this thesis. Green color indicates that the result is for trees, blue for tree-like graphs, and red for planar graphs. A continuous arrow from A to B means that A is used as a tool in the proof of B. A dotted arrow from A to B means that B generalizes the statement of A to tree-like graphs.

Chapter 2

Preliminaries and Notation

This chapter introduces the basic notation used throughout the thesis and states some general knowledge about graphs and algorithms. First, Section 2.1 introduces the basic notation concerning graphs and cuts. In particular, the notation for bisections and k-sections is introduced in Section 2.1. Afterward, Section 2.2 states some facts about graphs. Some basic facts stated there will be used in the following chapters without further mentioning them and others are stated to be able to refer to them later. Tree decompositions are introduced in Section 2.3, where also the properties (T1), (T2), (T3), and (T3') are stated, which are referred to by these names throughout the entire thesis. Preliminaries concerning algorithms are introduced in Section 2.4, which first discusses algorithms for graphs and then algorithms receiving a tree decomposition as input.

2.1 Basic Definitions

Before starting with the notation involving graphs, a few basic definitions are presented for the sake of completeness. Let $\mathbb{N} = \{1, 2, 3, \ldots\}$ be the set of *natural numbers* and define $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. For $n \in \mathbb{N}$, let $[n] = \{1, 2, \ldots, n\}$. Sometimes, it will be useful to write $[0]$, which is defined as the empty set. As usual, for a real x, the largest integer i with $i \leq x$ is denoted by $\lfloor x \rfloor$, and the smallest integer i with $i \geq x$ is denoted by $\lceil x \rceil$.

Graphs

In this thesis, all graphs are finite, undirected, and do neither have vertex nor edge weights. So a *graph* $G = (V, E)$ consists of a finite vertex set and an edge set $E \subseteq \binom{V}{2} := \{\{v, w\} \colon v \in V,\ w \in V,\ v \neq w\}$. Except for a few occurrences in iterative procedures, the vertex set of a graph is always assumed to be non-empty. For a graph G, denote by $V(G)$ its *vertex set* and by $E(G)$ its *edge set.*

Consider a graph $G = (V, E)$. An edge $e \in E$ and a vertex $v \in V$ are called *incident* if $v \in e$. For two vertices $v, w \in V$, the vertex w is a *neighbor* of v if $\{v, w\} \in E$. If v is a neighbor of w, we also say that v and w are *adjacent.* The set of neighbors of v is called the *neighborhood* of v and is denoted by $N_G(v)$. For a vertex $v \in V$, let $\deg_G(v)$ denote its *degree*, which is defined as the number of edges in E that are incident to v or, equivalently, the number of neighbors of v. Sometimes, when it is clear from the context to which graph the degree refers, then $\deg(v)$ is used instead of $\deg_G(v)$. A vertex $v \in V$ with $\deg_G(v) = 0$

is called an *isolated vertex* and a vertex $v \in V$ with $\deg_G(v) = 1$ is called a *leaf of* G. The *maximum* and *minimum degree* of G are defined as $\Delta(G) := \max\{\deg_G(v)\colon\ v \in V\}$ and $\delta(G) := \min\{\deg_G(v)\colon\ v \in V\}$, respectively. If there is a $d \in \mathbb{N}$ such that the graph G satisfies $\deg_G(v) = d$ for all $v \in V$, then G is called *d-regular*.

Often, subgraphs, that are formed by removing vertices or edges from another graph, will be studied. Consider again a graph $G = (V, E)$. A graph H is a *subgraph* of G if H satisfies $V(H) \subseteq V$ and $E(H) \subseteq E$ and H itself is a graph. If H is a subgraph of G, then we write $H \subseteq G$. Induced subgraphs are subgraphs that, for a certain vertex set, contain all possible edges. More precisely, for a set $\emptyset \neq W \subseteq V$, let $G[W]$ be the graph with the vertex set W and the edge set $E \cap \{\{v, w\}\colon\ v, w \in W,\ v \neq w\}$. Then, $G[W]$ is a subgraph of G and is called the *subgraph of* G *induced by* W. Furthermore, for a set $W \subseteq V$ with $W \neq V$, the graph obtained from G by removing all vertices in W as well as their incident edges is denoted by $G-W$. Observe that $G - W$ is an induced subgraph of G, namely $G - W = G[V \setminus W]$. For sets $W = \{v\}$, we also write $G - v$ instead of $G - \{v\}$. Similarly, for removing edges, consider a set $F \subseteq E$ and denote by $G - F$ the graph obtained from G by removing each edge in F and, if $F = \{e\}$, then $G - e$ is used to abbreviate $G - \{e\}$. Note that, for an edge $e = \{v, w\}$, the notation $G - e$ can be interpreted in two ways. On the one hand, it can refer to removing the edge e and, on the other hand, it can refer to removing the vertices v and w. Throughout the thesis, the correct interpretation will be stated explicitly or will be clear from the context.

Cuts, Bisections, and k-Sections

Let $G = (V, E)$ be a graph and let $n := |V|$. A *cut* in G is a partition of V into several sets $B_1, \dots, B_k$ with $k \geq 2$ and is denoted by $(B_1, B_2, \dots, B_k)$. Here, the sets B_ℓ of a cut $(B_1, \dots, B_k)$ are allowed to be empty. For example, $(V, \emptyset)$ is a cut in G. Fix an integer $k \geq 2$ and a cut $(B_1, \dots, B_k)$ in G. An edge $\{v, w\} \in E$ is *cut* by $(B_1, \dots, B_k)$ if there are two distinct indices $\ell, \ell' \in [k]$ with $v \in B_\ell$ and $w \in B_{\ell'}$. The set of edges in E that are cut by $(B_1, \dots, B_k)$ is denoted by $E_G(B_1, \dots, B_k)$. Moreover, the *width* of $(B_1, \dots, B_k)$ is defined as the number of edges cut by $(B_1, \dots, B_k)$ and is denoted by $e_G(B_1, \dots, B_k)$. So, $e_G(B_1, \dots, B_k) := |E_G(B_1, \dots, B_k)|$. If the graph G is clear from the context, then $E_G(B_1, \dots, B_k)$ and $e_G(B_1, \dots, B_k)$ are also abbreviated to $E(B_1, \dots, B_k)$ and $e(B_1, \dots, B_k)$, respectively. If $k = 2$, then usually (B, W) is used instead of (B_1, B_2) to denote a cut into two pieces and the sets B and W are referred to as the *black set* and the *white set*, respectively. In the following, unless indicated otherwise, when a figure displays some cut (B', W') in a graph G', then the vertices in B' are colored black, the vertices in W' are colored white, and each edge of G' that is cut by (B', W') is colored red.

Let $k \geq 2$ be an integer and let $G = (V, E)$ be a graph. A *k-section* in G is a cut $(B_1, \dots, B_k)$ in G with $||B_\ell| - |B_{\ell'}|| \leq 1$ for all $\ell, \ell' \in [k]$, i. e., if n denotes the number of vertices of G, then $\left\lfloor \frac{n}{k} \right\rfloor \leq |B_\ell| \leq \left\lceil \frac{n}{k} \right\rceil$ for all $\ell \in [k]$. The *minimum k-section width of* G is defined as

$$\operatorname{MinSec}_k(G) := \min\left\{e_G(B_1, \dots, B_k)\colon\ (B_1, \dots, B_k) \text{ is a } k\text{-section in } G\right\}.$$

A k-section $(B_1, \dots, B_k)$ in G with $e_G(B_1, \dots, B_k) = \operatorname{MinSec}_k(G)$ is called a *minimum k-section* in G. A *bisection* in G is a 2-section in G, i. e., a cut (B, W) in G that satisfies $|B| = |W|$ if the number of vertices of G is even, and $||B| - |W|| = 1$ if the number of vertices of G is odd. Similarly, the *minimum bisection width* of G is defined as $\operatorname{MinBis}(G) := \operatorname{MinSec}_2(G)$ and a minimum 2-section in G is called a *minimum bisection* in G.

Consider a graph $G = (V, E)$ and let $n := |V|$. For $m \in [n]$, an *m-cut* in G is a cut (B, W) with $|B| = m$. So, an $\left\lfloor \frac{n}{2} \right\rfloor$-cut in G is a bisection in G. Approximate cuts relax this size constraint in the following way. For $m \in [n]$, the cut (B, W) is called a *simple approximate m-cut* in G if $\frac{1}{2}m < |B| \leq m$. Furthermore, for $0 \leq c < 1$ and $m \in [n]$, a cut (B, W) in G is a *c-approximate m-cut* in G if $cm \leq |B| \leq m$ and (B, W)

is a *strict c-approximate m-cut* in G if $cm < |B| \leq m$. Whenever the precise values of c and m do not matter, we omit them and use the term *approximate cut*. Sometimes, the term *exact m-cut* is used to refer to an m-cut in order to distinguish it from an approximate m-cut. The term *exact cut* is used to refer to an exact m-cut, when no specific value for m is given, but the size of the black set is specified by an input parameter.

Special Graphs and Paths

For an integer $n \geq 2$, the *complete graph* on n vertices is denoted by K_n and defined as the graph with vertex set $[n]$ and edge set $\binom{[n]}{2}$. For two integers $n \geq 1$ and $m \geq 1$, the *complete bipartite graph* with partition classes of size n and m is denoted by $K_{n,m}$ and is defined as the graph with vertex set $[n+m]$ and edge set $\{\{v, w\}\colon\ v \in [n],\ w \in [n+m] \setminus [n]\}$. For an integer $n \geq 2$, the *star* on n vertices is defined as the complete bipartite graph $K_{1,n-1}$. If $n \geq 3$, then the unique vertex v of the star $K_{1,n-1}$ with $\deg(v) \geq 2$ is called the *center vertex* of the star. Let $P_0 := (\{0\}, \emptyset)$ and, for $n \in \mathbb{N}$, let P_n be the graph with vertex set $[n] \cup \{0\}$ where two distinct vertices v and w are adjacent if and only if $|v - w| \leq 1$. For $n \in \mathbb{N}_0$, the graph P_n is called the *path of length n*. Observe that P_n contains $n+1$ vertices and n edges. For an integer $n \geq 3$, the *cycle* of length n is denoted by C_n and defined as the graph obtained from P_n by removing the vertex 0 and adding the edge $\{1, n\}$.

Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if there is a bijection $f : V \to V'$ such that $\{v, w\} \in E$ if and only if $\{f(v), f(w)\} \in E'$. Fix a graph $G = (V, E)$ and let $v, w \in V$. Then, G contains a *v,w-path* if there is a sequence $(u_0, u_1, \ldots, u_\ell)$ with $u_0 = v$, $u_\ell = w$, $u_h \in V$ for all $h \in [\ell] \cup \{0\}$ and $\{u_{h-1}, u_h\} \in E$ for all $h \in [\ell]$ such that $u_h \neq u_{h'}$ for all distinct $h, h' \in [\ell] \cup \{0\}$. Let $V_P := \{u_0, \ldots, u_\ell\}$ and $E_P := \{\{u_{h-1}, u_h\}\colon\ h \in [\ell]\}$. Then, the subgraph $P = (V_P, E_P)$ of G is called a *path* of length ℓ in G and is denoted by $(u_0, u_1, \ldots, u_\ell)$. The vertices $u_0 = v$ and $u_\ell = w$ are called the *ends* of P. Observe that P is isomorphic to P_ℓ and that v and w are only leaves of P if $\ell \geq 1$. If P is a v,w-path in G, then we say that *P joins v to w*. For $h, h' \in [\ell] \cup \{0\}$ with $h \leq h'$, the path $(u_h, \ldots, u_{h'})$ is called the *subpath* of P that joins u_h to $u_{h'}$. Similarly, for $\ell \geq 3$, $(u_1, \ldots, u_\ell)$ is a *cycle of length ℓ* in G when $u_h \in V$ for all $h \in [\ell]$, $\{u_h, u_{h+1}\} \in E$ for all $h \in [\ell - 1]$, and $\{u_\ell, u_1\} \in E$ as well as that $u_h \neq u_{h'}$ for all distinct $h, h' \in [\ell]$. Observe that, in this case, G contains a subgraph with vertex set $\{u_1, \ldots, u_\ell\}$ that is isomorphic to the cycle C_ℓ. A vertex $v \in V$ *is on the path* $P = (u_0, \ldots, u_\ell)$ if there is an $h \in [\ell] \cup \{0\}$ with $v = u_h$. Similarly, an edge $e \in E$ is on the path $P = (u_0, \ldots, u_\ell)$ if there is an $h \in [\ell]$ with $\{u_{h-1}, u_h\} = e$. Two paths P and P' in G with $V(P) \cap V(P') = \emptyset$ are called *vertex-disjoint* and, if $E(P) \cap E(P') = \emptyset$, then P and P' are called *edge-disjoint*. Moreover, two paths $P = (u_0, u_1, \ldots, u_\ell)$ and $P' = (u'_0, u'_1 \ldots, u'_{\ell'})$ are *internally disjoint* if $u_h \notin V(P')$ for all $h \in [\ell - 1]$ and $u_{h'} \notin V(P)$ for all $h' \in [\ell' - 1]$. Observe that two internally disjoint paths may have one or two common ends. A walk is a concept that is less strict than a path. A sequence $(u_0, u_1, \ldots, u_\ell)$ of vertices is called a *walk* in G, if $u_h \in V$ for all $h \in [\ell] \cup \{0\}$ and $\{u_{h-1}, u_h\} \in E$ for all $h \in [\ell]$. Observe that every path is a walk but not vice versa as a walk may reuse a vertex. Similarly as for paths, u,v-walks, vertex-disjoint walks, edge-disjoint walks, and internally disjoint walks are defined. Moreover, if G contains a u,v-walk, then G also contains a u,v-path, as each walk that is not a path can be shortened to become a path. For two paths $P = (u_0, \ldots, u_\ell)$ and $P' = (u'_0, \ldots, u'_{\ell'})$ with $u_\ell = u'_0$, the *walk obtained by glueing P and P' together* is the sequence $(u_0, \ldots, u_\ell, u'_1, \ldots, u'_{\ell'})$. Observe that $(u_0, \ldots, u_\ell, u'_1, \ldots, u'_{\ell'})$ is not necessarily a path and, hence, the concept of walks is useful.

Miscellaneous

Let $G = (V, E)$ be a graph. Then, $G = (V, E)$ is *connected* if for all $v, w \in V$ there is a v,w-path in G. A maximal connected subgraph of G, i. e., a subgraph $H \subseteq G$ such that there is no connected subgraph $H' \subseteq G$ with $H \subseteq H'$ and $H \neq H'$, is called a *component* of G. Moreover, G is called *acyclic* if G does not contain a subgraph that is isomorphic to C_ℓ for all integers $\ell \geq 3$. A *forest* is defined as an acyclic graph and a *tree* is a connected and acyclic graph. Here, the symbol T is usually used to denote a graph that is a tree.

Roughly speaking, to subdivide an edge e means to insert a new vertex on e, and to contract an edge e means to merge its vertices together. More precisely, consider a graph $G = (V, E)$ and let $e = \{u, v\}$ be an edge of G. *Subdividing* e means to remove e from G, to insert a new vertex w, and to insert the edges $\{u, w\}$ and $\{v, w\}$. If G' is a graph that is obtained from G by successively subdividing edges, then G' is called a *subdivision* of G. To *contract* e means to remove u and v from G and to insert a new vertex w that is adjacent to each vertex in $N_G(u) \cup N_G(v)$. If G' is isomorphic to a graph that is obtained from a subgraph of G by successively contracting edges, then G *contains* G' *as a minor*. Let $G' = (V', E')$ be a graph that is obtained from G by successively contracting edges. Then one can partition V into sets M_x with $x \in V'$ such that the following properties are satisfied:

- For every $x \in V'$, the set M_x is non-empty.
- For every $x \in V'$, the graph $G[M_x]$ is connected.
- For every $x, x' \in V$, the graph G' contains the edge $\{x, x'\}$ if and only if G contains an edge $\{v, v'\}$ with $v \in M_x$ and $v' \in M_{x'}$.

So, when contracting all edges in $G[M_x]$ for each $x \in V'$ and calling the resulting vertex x, the graph G' is obtained, see also Chapter 1.7 in [Die12].

Consider a graph $G = (V, E)$. A path $P \subseteq G$ is called a *longest* path in G if G contains no path P' with $|E(P')| > |E(P)|$. For two vertices $v, w \in V$, a path $P \subseteq G$ is called a *shortest* v,w-path in G if G contains no v,w-path P' with $|E(P')| < |E(P)|$. The *distance* of two vertices v and w in a graph G is the length of a shortest v,w-path in G and is denoted by $\operatorname{dist}_G(x, y)$. For a connected graph, the *diameter* is defined as

$$\operatorname{diam}(G) = \max\{\operatorname{dist}(x, y) \colon\ x, y \in V(G)\}.$$

Observe that the diameter of a tree T is the length of a longest path in the tree as, for $v, w \in V(T)$, every v,w-path in T is a shortest v,w-path.

The symbol $\mathbb{R}$ denotes the set of real numbers and $\mathbb{R}_{>0}$ denotes the set of positive, real numbers. The following asymptotic notation is used. Let $f, g : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ be two functions. We write $f(x) = \mathcal{O}(g(x))$ if there are two constants $c \in \mathbb{R}_{>0}$ and $x_0 \in \mathbb{R}_{>0}$ such that $f(x) \leq cg(x)$ for all $x \geq x_0$. Moreover, we write $f(x) = \Omega(g(x))$ if there are two constants $c \in \mathbb{R}_{>0}$ and $x_0 \in \mathbb{R}_{>0}$ such that $f(x) \geq cg(x)$ for all $x \geq x_0$, i.e., $f(x) = \Omega(g(x))$ is equivalent to $g(x) = \mathcal{O}(f(x))$. Finally, we write $f(x) = \Theta(g(x))$ if $f(x) = \mathcal{O}(g(x))$ and $f(x) = \Omega(g(x))$.

2.2 Some Facts Concerning Graphs

Here, some basic knowledge about graphs is presented. In the following, most of these facts are used without further mentioning.

Proposition 2.1 (see Theorem 1.5.1 and Corollary 1.5.3 in [Die12]).
For every graph $T = (V, E)$ the following statements are equivalent:

(i) T is a tree.
(ii) T is connected and $|E| = |V| - 1$.
(iii) For every $v, w \in V$, there is exactly one v,w-path in T.

In particular, it follows that every tree $T = (V, E)$ satisfies $|E| = \mathcal{O}(|V|)$, which will be used often in the following. Consider a graph $G = (V, E)$. As every edge in E contributes 1 to the degree of exactly two vertices, $\sum_{v \in V} \deg_G(v) = 2|E|$. For trees and forests, the following corollary is obtained.

Corollary 2.2.

a) For every tree $T = (V, E)$, the following holds: $\sum_{v \in V} \deg_T(v) = 2|V| - 2$.

b) For every forest $G = (V, E)$, the following holds: $\sum_{v \in V} \deg_G(v) \leq 2|V| - 2$.

Two Graphs and their Minimum Bisection Width

Here, two graphs are introduced, namely perfect ternary trees and grids. Moreover, bounds on their minimum bisection width are stated. To define the former one, the standard terminology for rooted trees is used, which is as follows. A *rooted tree* is a tree, where one vertex has been designated the *root*. Consider a tree T with root r and let x and y be two vertices in T. Then, y is called a *descendant* of x if x is on the unique r,y-path in T. Note that x is a descendant of itself. The parent of a vertex $x \neq r$, denoted by $p(x)$, is the neighbor of x that is on the x,r-path. If y is the parent of x, then x is a *child* of y. In a rooted tree, a *leaf* is a vertex with no child. Observe that this differs slightly from the definition of a leaf in a graph as v is a leaf in the rooted tree $(\{v\}, \emptyset)$ but not in the graph $(\{v\}, \emptyset)$. The *subtree rooted in x* is the subgraph of T that is induced by all descendants of x. Let h, k be two integers with $k \geq 2$ and $h \geq 0$. A *k-ary* tree is a rooted tree $T = (V, E)$ with the property that every vertex $v \in V$ has at most k children. A 2-ary tree is also called a *binary* tree and a 3-ary tree is also called a *ternary* tree. A k-ary tree $T = (V, E)$ is called *full* if every vertex $v \in V$ has exactly k children or is a leaf of T. A full, k-ary tree T, where each leaf of T has distance h to the root, is called a *perfect k-ary tree of height h*. Clearly, a perfect k-ary tree of height 0 has 1 vertex, and a perfect k-ary tree of height 1 has $k + 1$ vertices. Using that a perfect k-ary tree of height h is obtained by taking k vertex disjoint perfect k-ary trees of height $h - 1$, adding a new vertex r that is the root, and joining r to each root of the k perfect k-ary trees of height $h - 1$, the next proposition follows by induction.

Proposition 2.3.
For every $h \in \mathbb{N}_0$ and every integer $k \geq 2$, a perfect k-ary tree of height h has $\frac{1}{k-1}\left(k^{h+1} - 1\right)$ vertices.

In [Sch13], the following bounds on the minimum bisection width in perfect ternary trees are derived.

Theorem 2.4 (see Corollary 4.12 in [Sch13]).
For every $h \in \mathbb{N}$, every perfect ternary tree T_h of height h satisfies

$$h - \log_3(h) \;\leq\; \mathrm{MinBis}(T_h) \;\leq\; h - \log_3(h) + 3.$$

The second graph introduced in this paragraph is the *grid*. For each $k \in \mathbb{N}$, the graph $G_k = (V_k, E_k)$ defined by

$$V_k := \{(i, j)\colon\ i \in [k],\ j \in [k]\} \qquad \text{and}$$

$$E_k := \left\{ \{(i, j), (i', j')\} \in \binom{V_k}{2}\colon\ \ |i - i'| + |j - j'| = 1 \right\}$$

is called the *$k \times k$ grid*. In drawings of G_k, unless stated otherwise, for all $i, j \in [k]$, the vertex (i, j) is drawn at the point (i, j) of a coordinate system, see Figure 2.1. In G_k, the vertex set $C_i = \{(i, j)\colon\ j \in [k]\}$ is called the i^{th} *column* for $i \in [k]$ and the vertex set $R_j = \{(i, j)\colon\ i \in [k]\}$ is called the j^{th} *row*. An

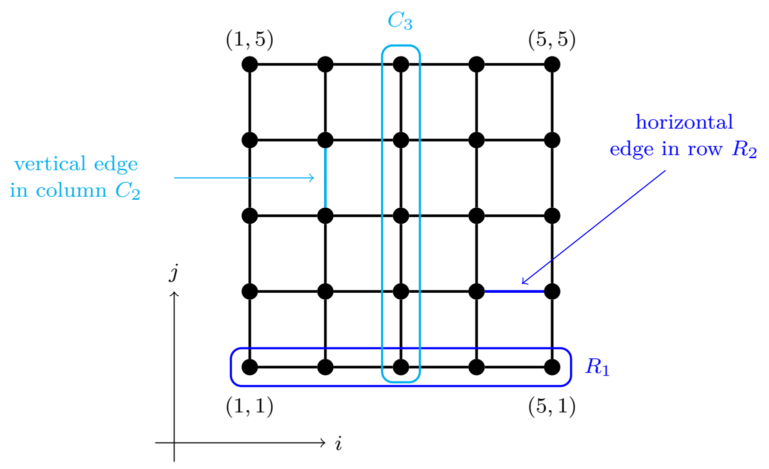


Figure 2.1: A 5×5 grid and its usual embedding.

edge $\{x,y\} \in E_k$ is called a *vertical edge in column* C_i if $x \in C_i$ and $y \in C_i$. An edge $\{x,y\} \in E_k$ is called a *horizontal edge in row* R_j if $x \in R_j$ and $y \in R_j$. Observe that each edge in E_k is either horizontal or vertical.

When the names of the vertices in G_k are not relevant, we often use the expression "a $k\times k$ grid" in order to refer to a graph isomorphic to the $k\times k$ grid G_k as defined above. We use the expression "$k\times k$ grid" to refer to the graph with the vertices and edges as defined above. A graph G isomorphic to the $k\times k$ grid for some $k \in \mathbb{N}$ is also called a *square grid*. Sometimes, non-square grids are used. More precisely, for two integers $k, k' \in \mathbb{N}$ the $k\times k'$ grid is defined to be the graph $G_{k,k'} = (V_{k,k'}, E_{k,k'})$ with

$$V_{k,k'} := \{(i,j) \colon\ i \in [k], j \in [k']\} \qquad \text{and}$$

$$E_{k,k'} := \left\{ \{(i,j),(i',j')\} \in \binom{V_{k,k'}}{2} \colon\ |i-i'| + |j-j'| = 1 \right\}.$$

Roughly speaking, square grids are well-connected, meaning that many edges need to be removed in order to cut off a linear fraction of the vertices. This is made precise by the following results.

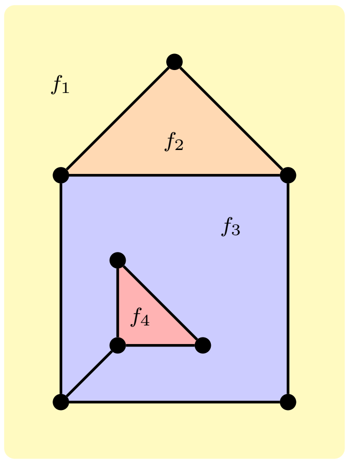
Lemma 2.5 (Theorem 6 in [LT79]).
For every $k \in \mathbb{N}$ and every $0 < \beta < \frac{1}{2}$, the following holds. Denote by $G_k = (V_k, E_k)$ the $k\times k$ grid and define $n := |V_k| = k^2$. If (B,W) is a cut in G_k with $\beta n \le |B| \le \frac{1}{2}n$, then $e_{G_k}(B,W) \ge k \cdot \min\left\{\frac{1}{2}, \sqrt{\beta}\right\}$.

Whereas the proof of the previous lemma is short and by a simple combinatorial argument, the following, stronger result has a long and involved proof.

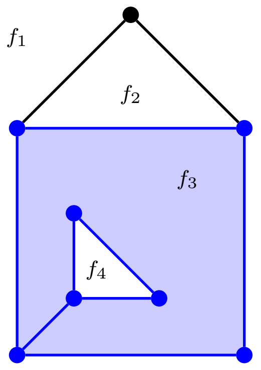
Lemma 2.6 (edge isoperimetric inequalities, see [BL91]).
For every $k \in \mathbb{N}$ the following holds. If (B,W) is a cut in the $k\times k$ grid G_k with $\frac{1}{4}k^2 \le |B| \le \frac{3}{4}k^2$, then $e_{G_k}(B,W) \ge k$.

Fix an integer $k \ge 2$. With $\beta = \frac{1}{4}$ Lemma 2.5 implies that every bisection (B,W) in the $k\times k$ grid cuts at least $\frac{1}{2}k$ edges. Lemma 2.6 is stronger and implies that every bisection (B,W) in the $k\times k$ grid cuts at least k edges. Another simple proof for a lower bound on the minimum bisection width of the square grid is also found in Chapter 1.9.1 in [Lei92].

Corollary 2.7.
For every integer $k \ge 2$ the $k\times k$ grid G_k satisfies $\mathrm{MinBis}(G_k) \ge k$.



a) Example of the faces of a plane graph.



b) The vertices and edges on the boundary of the face f_3 are colored blue.

Figure 2.2: A plane graph and its faces.

Planar Graphs

Roughly speaking, a drawing of a graph G is called *planar* if no two edges of G cross and if two edges touch, then they touch in a common vertex. A graph G is called *planar* if G admits a planar drawing. For a more formal definition of a planar graph as well as the following definitions, the reader is referred to Chapter 4 in [Die12]. A drawing of a graph is also called an *embedding in the plane*, or short an *embedding* as here no other surfaces are considered. Whenever an embedding or a drawing of a graph is considered here, we assume that it is planar. A *plane* graph is a graph G together with a drawing of G. Consider a plane graph G. Then, the drawing of G divides the plane into several regions, which are called *faces* of G, see Figure 2.2a). The following results for planar graphs are well-known.

Theorem 2.8 (Euler's Formula, see Theorem 4.2.9 in [Die12]).
Every connected plane graph with n vertices, m edges, and f faces satisfies $n - m + f = 2$.

Corollary 2.9 (Corollary 4.2.10 in [Die12]).
Every planar graph $G = (V, E)$ with $|V| \geq 3$ satisfies $|E| \leq 3|V| - 6$.

Theorem 2.10 (Kuratowski's Theorem, see Theorem 4.4.6 in [Die12]).
A graph G is planar if and only if G does not contain a subgraph that is isomorphic to a subdivision of K_5 or a subdivision of $K_{3,3}$.

Consider a plane graph $G = (V, E)$, i. e., some embedding of G is given. The boundary of a face f of G is the set of points p in the drawing of G such that every open ball around p contains a point in f and a point $\neq x$ that is not in f. Here, points that represent a vertex of G or belong to a polygon representing an edge of G do not belong to any face of G. For a vertex $v \in V$ that is embedded in the boundary of a face f, we say that v *is on the boundary of* f and for an edge $e \in E$ that is embedded in the boundary of a face f, we say that e *is on the boundary of* f. Figure 2.2b) gives an example for the boundary of a face.

2.3 Tree Decompositions

In this section, some preliminaries concerning tree decompositions are presented.

Definition 2.11.
Let $G = (V, E)$ be a graph. A pair $(T, \mathcal{X})$ with $\mathcal{X} = (X^i)_{i \in V(T)}$ is a *tree decomposition* of G if T is a tree, $X^i \subseteq V$ for every $i \in V(T)$, and the following three properties are satisfied.

(T1) For every $v \in V$, there is some $i \in V(T)$ with $v \in X^i$.

(T2) For every $e \in E$, there is some $i \in V(T)$ with $e \subseteq X^i$.

(T3) For all $i, j, h \in V(T)$, if h is on the (unique) i,j-path in T, then $X^i \cap X^j \subseteq X^h$.

The *width* of a tree decomposition $(T, \mathcal{X})$ is $\max\left\{|X^i| - 1\colon\ i \in V(T)\right\}$. The *tree-width* $\operatorname{tw}(G)$ of a graph G is the smallest integer t such that G allows a tree decomposition of width t.

Consider a tree decomposition $(T, \mathcal{X})$ with $\mathcal{X} = (X^i)_{i \in V(T)}$ of a graph G. To easily distinguish the vertices of G from the vertices of T, the vertices in $V(T)$ are called *nodes* in the following. Furthermore, for $i \in V(T)$, the set X^i is referred to as the *cluster of* i in $(T, \mathcal{X})$. It is easy to show that (T3) is equivalent to the following condition, see also Section 2 in [Bod98]:

(T3') For every $v \in V$, the graph $T[I_v]$ with $I_v = \{i \in V(T)\colon\ v \in X^i\}$ is connected.

In the following, (T1), (T2), (T3), and (T3') always refer to the properties defined here.

Every graph $G = (V, E)$ has a tree decomposition. For example, let $T = (\{i\}, \emptyset)$ and $X^i = V$, then $(T, \mathcal{X})$ where $\mathcal{X}$ consists only of X^i is a tree decomposition of G. It follows that $\operatorname{tw}(G) \le n - 1$ for every graph G on n vertices. If a graph G allows a tree decomposition of width 0, then $E(G) = \emptyset$. Next, it is shown that every tree $\tilde{T}$ with at least two vertices has tree-width 1. Let $\tilde{T} = (\tilde{V}, \tilde{E})$ be a tree with $|\tilde{V}| \ge 2$. To construct a tree decomposition of $\tilde{T}$, let T be the tree obtained from $\tilde{T}$ by subdividing each edge of $\tilde{T}$ once. For each $e \in \tilde{E}$, denote by i_e the vertex used to subdivide e. For each $v \in \tilde{V}$, define $X^v = \{v\}$ and, for each $e = \{v, w\} \in \tilde{E}$, define $X^{i_e} = \{v, w\}$. Let $\mathcal{X} = (X^i)_{i \in V(T)}$. Clearly, $(T, \mathcal{X})$ satisfies (T1) and (T2). To see that (T3') is satisfied, let $v \in \tilde{V}$, then $v \in X^i$ if and only if $i = v$ or $i = i_e$ for an edge e that is incident to v in $\tilde{T}$. Hence, for each $v \in \tilde{V}$, the set $\{j \in V(T)\colon\ v \in X^j\} = \{v\} \cup \{i_e\colon\ e \in \tilde{E}, v \in e\}$ induces a connected subgraph of T, namely a star or an isolated vertex. Consequently, $(T, \mathcal{X})$ is a tree decomposition of $\tilde{T}$ and $\operatorname{tw}(\tilde{T}) \le 1$. As $\tilde{T}$ contains at least one edge, $\operatorname{tw}(\tilde{T})$ must be at least 1 due to (T2).

Next, a tree decomposition of the square grid is presented. Fix an integer $k \ge 2$ and let $\tilde{G} = (\tilde{V}, \tilde{E})$ be the $k \times k$ grid. Recall that $\tilde{V} = \left\{(\tilde{i}, \tilde{j})\colon\ \tilde{i} \in [k], \tilde{j} \in [k]\right\}$. For each $i \in [k-1]$ and each $j \in [k]$ define

$$X^{(i-1)k+j} := \left\{(i, \tilde{j}) \in \tilde{V}\colon\ \tilde{j} \ge j\right\} \cup \left\{(i+1, \tilde{j}) \in \tilde{V}\colon\ \tilde{j} \le j\right\},$$

see Figure 2.3 for a visualization. Let T be the path obtained from $P_{k(k-1)}$ by removing the node 0 and let $\mathcal{X} = (X^h)_{h \in [k(k-1)]}$. For all $\tilde{i}, \tilde{j} \in [k]$,

$$(\tilde{i}, \tilde{j}) \in X^h \qquad \Leftrightarrow \qquad h \in \left\{(\tilde{i} - 2)k + \tilde{j}, \dots, (\tilde{i} - 1)k + \tilde{j}\right\} \cap [k(k-1)].$$

It follows that (T1) and (T3') are satisfied. To see that (T2) is satisfied, let $e \in \tilde{E}$. If $e = \{(\tilde{i}, \tilde{j}), (\tilde{i}+1, \tilde{j})\}$ for some $\tilde{i} \in [k-1]$ and $\tilde{j} \in [k]$, then $e \subseteq X^h$ for $h = (\tilde{i} - 1)k + \tilde{j}$. Otherwise, $e = \{(\tilde{i}, \tilde{j}), (\tilde{i}, \tilde{j}+1)\}$ for some $\tilde{i} \in [k]$ and $\tilde{j} \in [k-1]$ and $e \subseteq X^h$ for $h = (\tilde{i} - 1)k + \tilde{j}$. Consequently, $(T, \mathcal{X})$ satisfies (T2) and $(T, \mathcal{X})$ is a tree decomposition of $\tilde{G}$ of width k. Hence, $\operatorname{tw}(\tilde{G}) \le k$. A matching lower bound is stated in Lemma 88 in [Bod98] or Exercise 21 in Chapter 12 of [Die12]. This yields the next proposition.

Proposition 2.12.

a) Each tree T on at least 2 vertices satisfies $\operatorname{tw}(T) = 1$.

b) For each integer $k \ge 2$, the $k \times k$ grid G_k satisfies $\operatorname{tw}(G_k) = k$.

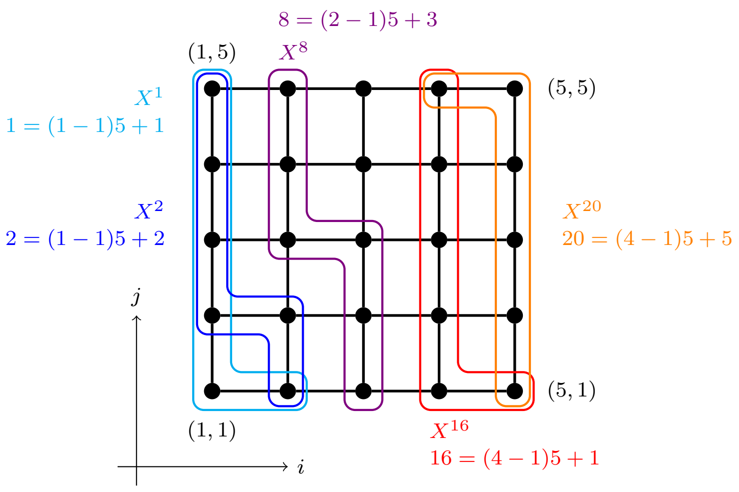


Figure 2.3: A few clusters of a tree decomposition of the 5×5 grid of width 5.

A tree decomposition $(T,\mathcal{X})$ of a graph G is a *path decomposition* of G if T is a path. The tree decomposition of the square grid that was presented above is a path decomposition. The width of a path decomposition is defined as the width of a tree decomposition. The *path-width* of a graph G is the smallest integer t such that G allows a path decomposition of width t and is denoted by $\mathrm{pw}(G)$. Clearly, all graphs G satisfy $\mathrm{pw}(G) \geq \mathrm{tw}(G)$.

For planar graphs, the following bound on the tree-width is known. Its proof can be found in [Bod98]. There, Corollary 23 states that the path-width of a planar graph on n vertices is at most $\mathcal{O}(\sqrt{n})$. Using that $\mathrm{tw}(G) \leq \mathrm{pw}(G)$ for all graphs G implies the next proposition.

Proposition 2.13.
Every planar graph G on n vertices satisfies $\mathrm{tw}(G) = \mathcal{O}(\sqrt{n})$.

Consider a graph $G = (V,E)$, a tree decomposition $(T,\mathcal{X})$ with $\mathcal{X} = (X^i)_{i\in V(T)}$, and a graph $H \subseteq G$. A tree decomposition for H can be easily obtained from $(T,\mathcal{X})$ by deleting all vertices that are not in H. More precisely, for $i \in V(T)$, let $\tilde{X}^i = X^i \cap V(H)$. Then, it is easy to check that $(T,\tilde{\mathcal{X}})$ with $\tilde{\mathcal{X}} = (\tilde{X}^i)_{i\in V(T)}$ is a tree decomposition of H. The tree decomposition $(T,\tilde{\mathcal{X}})$ is called the *induced tree decomposition* of H with respect to $(T,\mathcal{X})$. Observing that the width of $(T,\tilde{\mathcal{X}})$ is at most the width of $(T,\mathcal{X})$ yields the next proposition.

Proposition 2.14.
Let G be a graph and let $(T,\mathcal{X})$ be a tree decomposition of G of width $t-1$. For every subgraph $H \subseteq G$, the induced tree decomposition of H with respect to $(T,\mathcal{X})$ is a tree decomposition of H of width at most $t-1$. Furthermore, $\mathrm{tw}(H) \leq \mathrm{tw}(G)$.

Similarly to the construction for subgraphs, a tree decomposition of a minor can be constructed, see also Lemma 12.3.3 and Proposition 12.3.6 in [Die12]. The following proposition is obtained.

Proposition 2.15.
For all graphs H and G such that G contains H as a minor, $\mathrm{tw}(H) \leq \mathrm{tw}(G)$.

Often, when constructing a cut in a tree $\tilde{T}$, the vertex set of $\tilde{T}$ is partitioned by removing all edges incident to a vertex $v \in V(\tilde{T})$ and considering the vertex sets of the resulting components. Then, a cut in $\tilde{T}$ of width at most $\Delta(\tilde{T})$ is obtained when combining these vertex sets in an arbitrary way. This idea can be generalized by considering clusters of a tree decomposition, as done in the next lemma. It uses the following notation: Consider a graph $G = (V, E)$ and a tree decomposition $(T, \mathcal{X})$ of G. For each node i in T define

$$E_G(i) = \{e \in E : e \cap X^i \neq \emptyset\} \qquad \text{and} \qquad e_G(i) = |E_G(i)|,$$

where X^i denotes the cluster of i in $(T, \mathcal{X})$. Observe that, when $t-1$ denotes the width of $(T, \mathcal{X})$, then $e_G(i) \leq |X^i|\Delta(G) \leq t\Delta(G)$ for every $i \in V(T)$. We say that two subgraphs $H_1 \subseteq G$ and $H_2 \subseteq G$ are *disjoint parts* of G if $V(H_1) \cap V(H_2) = \emptyset$ and there is no edge $e = \{v, w\}$ in G with $v \in V(H_1)$ and $w \in V(H_2)$. Note that, if G is not connected, then two distinct components of G are disjoint parts of G, but the subgraph H_i for $i \in \{1, 2\}$ in the definition of disjoint parts does not have to be connected. The next lemma says that, if the vertices in X^i or the edges in $E_G(i)$ are removed for some $i \in V(T)$, then the graph G splits into several disjoint parts. So, these disjoint parts can be combined in an arbitrary way to obtain a cut in G of width at most $e_G(i) \leq t\Delta(G)$. The lemma is a widely known fact about tree decompositions, similar statements are Fact 10.13 and Fact 10.14 in [KT06] or Corollary 1.8 in [Ree97].

Lemma 2.16.
Let $G = (V, E)$ be an arbitrary graph and let $(T, \mathcal{X})$ be a tree decomposition of G with $\mathcal{X} = (X^j)_{j \in V(T)}$. Fix an arbitrary node $i \in V(T)$, let $k := \deg_T(i)$, and denote by $i_1, i_2, \ldots, i_k$ the neighbors of i in T. For $\ell \in [k]$, let V_ℓ^T be the node set of the component of $T - i$ that contains i_ℓ and define $V_\ell := \bigcup_{j \in V_\ell^T} X^j \setminus X^i$.

a) Removing the vertices in X^i from G decomposes G into k disjoint parts, which are $G[V_1], \ldots, G[V_k]$.
b) Removing the edges in $E_G(i)$ from G decomposes G into $k + |X^i|$ disjoint parts, which are $(\{v\}, \emptyset)$ for every $v \in X^i$ and $G[V_\ell]$ for every $\ell \in [k]$.

Proof. Let $G = (V, E)$, $(T, \mathcal{X})$ with $\mathcal{X} = (X^j)_{j \in V(T)}$, i, k, $i_1, \ldots, i_k$, as well as V_ℓ^T and V_ℓ for each $\ell \in [k]$ be as in the statement.

a) For each $v \in V$ let $I_v := \{j \in V(T) \colon v \in X^j\}$, which is the same as in (T3'). Due to (T1), it follows that

$$\bigcup_{\ell \in [k]} V_\ell = \left(\bigcup_{j \in V(T) \setminus \{i\}} X^j \right) \setminus X^i = V \setminus X^i.$$

Consider a vertex $v \in V$. If $v \in X^i$, then $v \notin V_\ell$ for every $\ell \in [k]$. Otherwise, $v \notin X^i$ and $I_v \subseteq V_\ell^T$ for a unique $\ell \in [k]$ as $T[I_v]$ is connected by (T3'), nonempty by (T1), and does not contain the node i. So $v \in V_\ell$ implies $v \notin V_{\ell'}$ for all $\ell' \neq \ell$ and therefore the sets $V_1, \ldots, V_k$ are a partition of $V \setminus X^i$.

It remains to show that, for all distinct $\ell_1, \ell_2 \in [k]$, there is no edge $\{v_1, v_2\} \in E$ with $v_1 \in V_{\ell_1}$ and $v_2 \in V_{\ell_2}$. Assume, for a contradiction, that there is such an edge in E. Property (T2) says that there is a node j^* with $v_1 \in X^{j^*}$ and $v_2 \in X^{j^*}$. So, $j^* \in I_{v_1} \cap I_{v_2}$. Also $j^* \neq i$ because $v_1 \notin X^i$. As argued before, $I_{v_1} \subseteq V_{\ell_1}^T$ and $I_{v_2} \subseteq V_{\ell_2}^T$, because $v_1 \notin X^i$ and $v_2 \notin X^i$. As $\ell_1 \neq \ell_2$, the trees $T[V_{\ell_1}^T]$ and $T[V_{\ell_2}^T]$ are different components of $T - i$. Hence, $V_{\ell_1}^T \cap V_{\ell_2}^T = \emptyset$ and therefore $I_{v_1} \cap I_{v_2} = \emptyset$, but this contradicts that $j^* \in I_{v_1} \cap I_{v_2}$.

b) Clearly, every vertex $v \in X^i$ is an isolated vertex in $G - E_G(i)$. So, it suffices to show that $G - X^i$ decomposes into the k disjoint parts $G[V_1]$, $G[V_2], \ldots, G[V_k]$, which is equivalent to Part a). ☐

The next proposition says that low tree-width implies that a graph does not have many edges.

Proposition 2.17 (Fact 1.10 in [Ree97]).
Every graph G on n vertices satisfies $|E(G)| \leq n\,\mathrm{tw}(G)$.

For a proof see Fact 1.10 in [Ree97].

2.4 Algorithms

2.4.1 Graphs

Let $G = (V, E)$ be a graph on n vertices. For algorithms receiving G as input, it is always assumed that $V(G) = [n]$ and that G is given by its adjacency lists. Note that then the number of vertices in G is also known. Consider an algorithm that receives G and possibly some integers as input, for example an algorithm that computes an m-cut in G. The algorithm runs in linear time if its running time is bounded by a function that is linear in the input size, which is $\Theta(|V(G)| + |E(G)|)$ as each edge corresponds to exactly two entries in the adjacency lists. Therefore, the following definition is used.

Definition 2.18.
The *size* of a graph $G = (V, E)$ is defined as $\|G\| := |V| + |E|$.

Most of the algorithms presented here will compute a cut (B, W) in the input graph G. Usually, the cut (B, W) will be returned as a list of the vertices in the set B, which is not sorted and does not contain any repetitions. It is easy to order the vertices in the set B in $\mathcal{O}(n)$ time with the counting sort algorithm, which is presented in the next lemma.

Lemma 2.19 ([Cor+09]).
A list of m items in $[n]$ can be sorted in $\mathcal{O}(n + m)$ time with the counting sort algorithm.

Basically, the idea is to put each item with value i in the i^{th} place of an array of length n. Then, the ordered list of all m items can be obtained by traversing the array once. For details, see Chapter 8.2 in [Cor+09]. So, when an algorithm uses the counting sort algorithm to sort the vertices in the black set of a cut, then a list of the vertices in the white set can be read off the array as well, because a vertex lies in the white set if and only if its entry in the array is undefined.

Proposition 2.20.
Consider a graph $G = (V, E)$ on n vertices with $V = [n]$ and a cut (B, W) in G. When given an unordered list of the vertices in B, a list of the vertices in W can be determined in $\mathcal{O}(n)$ time.

Let G be a graph on n vertices with $V = [n]$. The advantage of storing sets as unordered lists without repetitions is that computing the union of disjoint sets can be performed in $\mathcal{O}(1)$ time by concatenating the lists. Note that, when subsets of the vertices of a graph on n vertices are stored in binary arrays of length n, then it is necessary to traverse at least one of the arrays to compute the union of two sets. However, this approach also works when the subsets are not disjoint. Here, we will mostly work with disjoint subsets and therefore sets are stored as unordered lists unless indicated otherwise.

Consider a graph $G = (V, E)$ on n vertices. The assumption $V = [n]$ is natural for input graphs, see, for example, Chapter 4.1 in [SW11]. However, when considering an iterative procedure, where some algorithm is applied to the graph G and shall later be applied to a subgraph $G' \subseteq G$ on n' vertices, then $V' := V(G') = [n']$ is usually not satisfied. Still, the assumption $V = [n]$ cannot simply be ignored, as we will see in Section 2.4.2, where some procedures that rely on arrays, which are indexed with the vertices of G, are introduced. Weakening the assumption to $V' \subseteq [n]$ might not be enough, as initializing an array of length n might take too long when n' is much smaller than n. Therefore, whenever an algorithm is applied to a subgraph G' and it relies on the assumption that $V' = [n']$, we will set up a bijection f between V' and $[n']$. The bijection f is stored in two arrays F and F' of length n and n', respectively. For each vertex $v \in V'$, the entry $F[v]$ contains $f(v)$ and, for each vertex $v \in V \setminus V'$, the entry $F[v]$ may contain an arbitrary value. For each integer $s \in [n']$, the entry $F'[s]$ contains $f^{-1}(s)$, i. e., the vertex $v \in V'$ which is mapped to s by f. When advancing from G' to a second subgraph $G'' \subseteq G'$, the bijection f is modified to be a bijection between $V(G'')$ and $[n'']$, where n'' denotes the number of vertices of G''. This is necessary, because applying too many bijections increases the running time of converting back the vertex names in the current subgraph to vertex names in the original graph. In particular, one has to be careful when the number of iterations is not bounded by a constant. The next lemma formalizes these ideas and also presents a method to check whether a vertex $v \in V$ belongs to V' in constant time. Note that F' contains a list of the vertices in V' but, for a single vertex $v \in V$, it would take $\Omega(n')$ time to check whether $v \in V'$ by traversing F'.

Lemma 2.21.
Consider a graph $G = (V, E)$ on n vertices with $V = [n]$, a subgraph $G' = (V', E') \subseteq G$ on n' vertices, and another subgraph $G'' = (V'', E'') \subseteq G'$ on n'' vertices.

a) Given a list of the vertices in G', a bijection between the vertex set of G' and the set $[n']$, which is stored in two arrays of length n and n', can be set up in $\mathcal{O}(n)$ time.

b) A bijection between the vertex subset V' and $[n']$ as in a) allows to convert each vertex name in V' to the corresponding integer in $[n']$ in constant time and vice versa.

c) Given a bijection as in a), it is possible to check whether $v \in V(G)$ belongs to the subgraph G' in constant time.

d) Given a bijection between the vertices in G' and the set $[n']$ as well as a list of the vertices in G'', the bijection can be updated to a bijection between the vertices in G'' and the set $[n'']$ in $\mathcal{O}(n'')$ time.

Proof. Let $G = (V, E)$, $G' = (V', E')$, and $G'' = (V'', E'')$ be as in the statement and denote by n, n', and n'' their number of vertices, respectively.

a) First, the algorithm initializes two arrays F and F' of length n and n', respectively, with zeros. Then it traverses the list L' of vertices in V'. For the s^{th} entry v in L', the algorithm sets $F[v] = s$ and $F'[s] = v$, which means that the bijection maps $v \in V'$ to $s \in [n']$. All in all, this procedure takes $\mathcal{O}(n + n') = \mathcal{O}(n)$ time.

b) All necessary information is stored in the arrays of the bijection and accessing an entry of an array takes constant time.

c) Assume a bijection as in a) is stored in two arrays F and F'. Consider an arbitrary vertex $v \in V$. A vertex $v \in V$ is in V' if and only if there is an entry in F' that contains v. Furthermore, for each vertex $v \in V'$, the entry $F[v]$ is set to the index s with $F'[s] = v$. So, for an arbitrary $v \in V$, the algorithm does the following. First, it checks whether $F[v]$ is an integer in $[n']$. If not, then $v \notin V'$. If $F[v] = s \in [n']$, it checks whether $F'[s] = v$. If so, then $v \in V'$ and otherwise $v \notin V'$. Clearly, this procedure takes constant time.

d) Assume a bijection between V' and $[n']$ is stored in two arrays F and F' as in a), and a list L'' of the vertices in $G'' \subseteq G'$ is given. First, the algorithm initializes a new array F'' of length n'' with zeros. Then it traverses the list L'' and does the same as the algorithm in a) to obtain a bijection between V'' and $[n'']$ stored in the arrays F and F''. Note that this procedure works, as the algorithm does not read any information from the array F. Furthermore, the array F does not need to be initialized and, hence, the running time reduces to $\mathcal{O}(n'')$, compared to $\mathcal{O}(n + n'')$ time for the procedure described in a). □

In Part d), the array F' is not overwritten for the following reason. When working with the subgraph G', the assumption $V(G') = [n']$ will be used, which is feasible when a bijection as in Part a) has been set up. However, then the subgraph $G'' \subseteq G'$ will not be described with vertex names referring to the original vertex names in G, but with vertex names referring to the vertex set of G' after renaming the vertices of G' according to the bijection. Hence, it is natural to have a list of the vertices in G'', where each vertex is renamed according to the bijection between $V(G')$ and $[n']$. Then, Part b) allows the algorithm to convert back to the corresponding vertex names of the original graph G by using the array F'.

Here, the assumption $V(G) = [n]$ is often needed when working with a tree decomposition as usually it is necessary to compute the union or the intersection of clusters. Therefore, whenever a tree decomposition is involved, we are careful and explicitly discuss how to rename the vertices such that $V(G) = [n]$ is satisfied whenever a subroutine is called that receives a graph G, or maybe only a tree decomposition of G, as input. When working with a tree or a forest, we are less careful, as the next lemma says that renaming the vertices is quick.

Lemma 2.22.
Let $n_0, n \in \mathbb{N}$ be two integers with $n \leq n_0$ and let $G = (V, E)$ be a forest on n vertices with $V \subseteq [n_0]$. In $\mathcal{O}(n)$ time, the vertices of G can be renamed such that $V(G) = [n]$ and an array F' of length n can be set up such that, for $s \in [n]$ the entry of $F'[s]$ contains the vertex of G that was renamed to s.

Proof. Fix $n_0, n \in \mathbb{N}$ with $n \leq n_0$ and consider a forest $G = (V, E)$ on n vertices with $V \subseteq [n_0]$ that is represented by its adjacency lists. For the following running time estimation, it is assumed that reserving and freeing any amount of space in memory takes constant time. First, the algorithm determines the number of vertices of G, i. e., the number of adjacency lists, and the largest number $n'_0 \in [n_0]$ with $n'_0 \in V$, which takes $\mathcal{O}(n)$ time by traversing the adjacency lists of G. Then, the algorithm creates an integer array F' of length n and initializes it with zeros, which takes $\mathcal{O}(n)$ time. Moreover, it reserves space for an integer array F of length n'_0 but does not initialize the array F. So, setting up F takes constant time. The algorithm traverses the set V and, for the s^{th} vertex $v \in V$, it sets $F[v] = s$ and $F'[s] = v$, which takes $\mathcal{O}(n)$ time. Observe that now, if $v \in [n'_0]$ is a vertex of G, then $F[v]$ contains a number s such that $F'[s] = v$ and, if $v \in [n_0]$ is not a vertex of G, then either $F[v]$ does not contain an index in $[n]$ or $F'[F[v]]$ is not v. To rename the vertices, the algorithm traverses the adjacency lists of G and exchanges each entry v with $F[v]$. Finally the algorithm frees the space reserved for the array F. The array F' contains all information needed to convert back the names of the vertices in the graph G. □

Consider a tree T_0 on n_0 vertices and an algorithm that applies some procedure, which returns a vertex set and a subgraph T to which the same procedure is applied again and so on. As long as the procedure takes $\mathcal{O}(n)$ time for a tree T on n vertices, renaming the vertices of the current tree with Lemma 2.22 does not increase the asymptotic running time. Indeed, setting up the bijection takes $\mathcal{O}(n)$ time and converting the vertex names of the computed set back to the original names takes constant time per vertex and, hence, at most $\mathcal{O}(n)$ time in total. Thus, to estimate the asymptotic running time of the algorithm, the time needed for renaming the vertices can be neglected.

Other than setting up and modifying a bijection in each iteration, one can also reuse the arrays of the first iteration: Then, after each iteration all arrays need to be cleaned up. For example, if an array of length n initialized with zeros is needed, then each entry needs to be set back to zero after the first iteration. This does not require to traverse the entire array when the algorithm keeps track of the modified entries and sets them back to zero, or as usually only entries referring to vertices of the current subgraph are modified, the algorithm can traverse all corresponding entries. This additional step of cleaning up arrays requires at most as much time as the iteration itself. Then, the next procedure does not need to initialize the arrays and the weakened assumption $V(G') \subseteq [n]$ suffices.

Lemma 2.21 is also useful for computing induced subgraphs and subgraphs created by removing some set of vertices, as the following corollaries show.

Corollary 2.23.
Consider a graph $G = (V, E)$ on n vertices with $V = [n]$ and a vertex set $V' \subseteq V$. Let $n' := |V'|$. There is an algorithm that computes the adjacency lists of $G' := G[V']$ in $\mathcal{O}(\|G\|)$ time, when given the adjacency lists of G and a list of the vertices in V'. While doing so, it can set up a bijection or update an existing bijection between $V(G')$ and $[n']$ as in Lemma 2.21a).

Proof. Let $G = (V, E)$, n, V', and G' be as in the statement. Using the list of vertices of G', the algorithm sets up a bijection between $V(G')$ and $[n']$, which takes $\mathcal{O}(n)$ time by Lemma 2.21a). This also provides a method to check whether a vertex $v \in V$ is in V' in constant time by Lemma 2.21c). Then, the algorithm traverses the adjacency lists of G and creates a copy that contains only the entries that are relevant for the subgraph G'. More precisely, when an adjacency list of a vertex $v \in V$ is processed, then it is skipped if $v \notin V'$. Otherwise, $v \in V'$ and the algorithm traverses all entries w of the adjacency list of v and keeps the ones with $w \in V'$. This procedure takes $\mathcal{O}(n + \|G\|) = \mathcal{O}(\|G\|)$ time. □

Corollary 2.24.
Consider a graph $G = (V, E)$ on n vertices with $V = [n]$ and a vertex set $S \subseteq V$. Let $n' := n - |S|$. There is an algorithm that computes the adjacency lists of $G' := G - S$ in $\mathcal{O}(\|G\|)$ time, when given the adjacency lists of G and a list of the vertices in S. While doing so, it can set up a bijection or update an existing bijection between $V(G')$ and $[n']$ as in Lemma 2.21a).

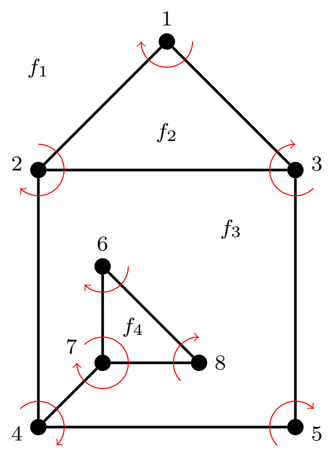
Proof. Let $G = (V, E)$, n, S, and G' be as in the statement. A list of the vertices in G' can be obtained in $\mathcal{O}(n)$ time by applying the same procedure as in Proposition 2.20. Then, the result follows from Corollary 2.23, as $G' := G - S = G[V \setminus S]$. □

The following lemma about computing the components of a graph is a basic fact, see Chapter 4.1 in [SW11].

Lemma 2.25.
Let $G = (V, E)$ be a graph. When traversing G with a depth-first search, one can compute

- *the number of components of G,*
- *for each component $\tilde{G}$ of G the number of vertices in $\tilde{G}$, and*
- *for each component $\tilde{G}$ of G a list of the vertices in $\tilde{G}$*

in $\mathcal{O}(\|G\|)$ time.



rotation systems

1	3, 2
2	1, 3, 4
3	5, 2, 1
4	2, 7, 5
5	4, 3
6	8, 7
7	6, 8, 4
8	7, 6

List of vertices in the boundaries

f_1	1, 3, 5, 4, 2
f_2	1, 2, 3
f_3	2, 4, 7, 6, 8, 7, 4, 5, 3
f_4	6, 7, 8

Figure 2.4: A plane graph G, its rotation systems, and the list of vertices on the boundary of each face, that is obtained when walking along the boundary. The red arrows indicate the circular ordering at each vertex.

Consider a graph $G = (V, E)$ that is represented by its adjacency lists. Recall that each edge $\{v, w\} \in E$ corresponds to two entries in the adjacency lists of G, namely the entry w in the adjacency list of v and the entry v in the adjacency list of w. For example, to delete the edge $\{v, w\}$ quickly, it is useful to have some connection between these two entries in the adjacency list of G. So, once one of the entries is found, the other one can be determined in constant time. This is achieved by adding pointers between the two entries representing one edge. More precisely, for the edge $\{v, w\}$, the entry w in the adjacency list of v contains additionally a pointer to the entry v in the adjacency list of w and vice versa. If this is the case for every entry in the adjacency lists of G, we say that the adjacency lists are *linked*. Linked adjacency lists can be set up in linear time, by first creating a list of the edges, which can be read off the adjacency lists, deleting all entries in the adjacency lists, and inserting each edge in its linked way.

When working with a planar graph, it is sometimes necessary to store an embedding of the graph in the plane. Consider an embedding of a planar graph G in the plane. All information that is necessary to reconstruct the combinatorial structure of the embedding of G, i. e., to retrieve the information which edges and vertices are on the boundary of which face, is the order in which the edges incident to each vertex need to be drawn. One simple way to store this information is to order the adjacency list of each vertex $x \in V(G)$ accordingly. The *circular ordering* of a vertex $x \in V(G)$ is the clockwise order of the edges incident to x in the considered embedding of G. When the adjacency list of x is ordered according to the circular ordering around x, it is called the *rotation system* of x, see Chapter 3.2 in [MT01] for details and Figure 2.4 for an example. Furthermore, when an algorithm receives a plane graph as input, it is assumed that it is represented by its rotation systems. Note that the embedding is not necessarily completely determined by the rotation systems for graphs that are not connected. For example, consider the graph consisting of a 3×3 grid and an isolated vertex x, then the rotation systems do not determine the face of the 3×3 grid in which the vertex x is embedded. When given a connected plane graph G, i. e., each adjacency list of G is a rotation system, then the combinatorial structure of the faces is uniquely determined and it can be obtained by walking along the boundary of each face. To do so, choose an arbitrary vertex v_1 and an arbitrary edge that joins v_1 to one of its neighbors, say v_2. The next vertex in the walk is the entry v_3 that follows v_1 in the rotation system of v_2 as the edge $\{v_2, v_3\}$ is embedded after the edge $\{v_2, v_1\}$ in the circular ordering of v_2. The vertices v_4, v_5, and so on are determined analogously. The procedure stops when the edge $\{v_1, v_2\}$ is traversed again in the same direction as in the beginning. To find all faces systematically, the algorithm marks the entries in the rotation systems that have been used already. More precisely, it marks the entry v_{i+1} in the adjacency list of v_i when using the edge $\{v_i, v_{i+1}\}$. Once all entries are marked, all faces are discovered. When the rotation systems are *linked*, where linked has the same meaning as for adjacency lists, this procedure takes $\mathcal{O}(\|G\|)$ time as each edge is traversed twice and it is not necessary to search for the entry v_{i+1} in the rotation system of v_i. If the rotation systems are not linked, but G is a bounded-degree graph, then the time to search for the entry v_{i+1} in the rotation system of v_i is constant and, hence, the running time is also $\mathcal{O}(\|G\|)$. Due to Corollary 2.9, the running time $\mathcal{O}(\|G\|)$ simplifies to $\mathcal{O}(n)$, where n denotes the number of vertices of G.

Lemma 2.26.
Given a connected plane graph G on n vertices with bounded degree as input, one can determine a list of the faces of G as well as, for each face f of G, a list of the vertices on the boundary of f in $\mathcal{O}(n)$ time.

Consider a connected plane graph G and fix a vertex $v \in V(G)$. If, for each face f of G, a list of the vertices on the boundary of f is known, then the algorithm can traverse all these lists to create a list of the faces of G that contain x in their boundary. Since the lists containing the vertices on the boundary of each face can be computed in $\mathcal{O}(\|G\|)$ time, traversing them requires $\mathcal{O}(\|G\|)$ time. As this procedure can be applied simultaneously for all vertices of G, the following lemma is obtained.

Lemma 2.27.
Given a connected plane graph $G = (V, E)$ on n vertices with bounded degree as input, one can compute, for each vertex $v \in V$, a list of the faces of G that contain v in their boundary in $\mathcal{O}(n)$ time.

There are algorithms that, when given an arbitrary planar graph G, compute a plane embedding of G. The fastest such algorithms run in linear time. The first such algorithm is due to Hopcroft and Tarjan [HT74]. A different method is described by Lempel, Even, and Cederbaum [LEC67]. Both methods heavily depend on the underlying data structure to store the part of the graph that is already examined, see also [CW90] for a comparison.

Theorem 2.28.
There is an algorithm that, when given a graph G on n vertices, computes a plane embedding of G or returns that G is not planar in $\mathcal{O}(n)$ time.

2.4.2 Tree Decompositions

In addition to algorithms working with graphs, also algorithms working with tree decompositions are presented here. In general, determining the tree-width of a graph is NP-hard, see Theorem 1 in [Bod98] and [ACP87]. For fixed $t \in \mathbb{N}$, there is an algorithm that, when given a graph G with $\text{tw}(G) \leq t$ as input, computes a tree decomposition of G of width at most t in linear time [Bod96]. However, the running time is already too long for practical purposes even if t is very small. Moreover, there is an algorithm that, when given a graph G on n vertices as input, computes a tree decomposition of G of width $\mathcal{O}\left(\text{tw}(G)\sqrt{\log(\text{tw}(G))}\right)$ in time polynomial in n and $\text{tw}(G)$, see Theorem 6.4 in [FHL08]. For planar graphs G on n vertices, there is an algorithm that approximates the tree-width of G within a factor of 1.5 in $\mathcal{O}(n^3)$ time [GT08], and relies on the relationship between branch-width and tree-width. It is open, whether computing a tree decomposition of minimum width of a planar graph is NP-hard.

Therefore, the algorithms presented here, that work with tree-like graphs, receive a tree decomposition of small width as input. The format is as follows. Consider a tree decomposition $(T, \mathcal{X})$ with $\mathcal{X} = (X^i)_{i \in V(T)}$ and denote by n_T the number of nodes of T. Similar to graphs, it is assumed that T is given by its adjacency lists and that the node set of T is $[n_T]$. Furthermore, it is assumed that each cluster in $\mathcal{X}$ is given as an unordered list of (distinct) vertices and each node $i \in V(T)$ has a pointer pointing to X^i. So, to store the tree decomposition $(T, \mathcal{X})$, first, the tree T itself needs to be stored, which takes $\|T\| = \mathcal{O}(|V(T)|)$ space and, second, each non-empty cluster needs to be stored, which takes $\sum_{i \in V(T)} |X^i|$ space. Taking into account that many clusters in $\mathcal{X}$ could be empty, it becomes clear that in the following definition both terms are needed.

Definition 2.29.
The *size* of a tree decomposition $(T, \mathcal{X})$ with $\mathcal{X} = (X^i)_{i \in V(T)}$ is defined as

$$\|(T, \mathcal{X})\| \ := \ |V(T)| + \sum_{i \in V(T)} \left|X^i\right|.$$

Note that, for every graph G on n vertices, every tree decomposition $(T, \mathcal{X})$ of G satisfies $n = \mathcal{O}(\|(T, \mathcal{X})\|)$ as every vertex of G needs to be in at least one of the clusters in $\mathcal{X}$ by (T1). This fact will be used in the following without explaining it further.

Most of the algorithms described here that use a tree decomposition are based on a depth-first search in the decomposition tree. Consider a tree decomposition $(T, \mathcal{X})$ where $V(T) = [n_T]$ for some integer n_T. While traversing T with a depth-first search, two arrays D and π, that are each of length n_T and store the following additional information, can be created. The array D is used to store the *discovery time* of each node and the array π is used to store the *predecessor* of each node. That is, if T was rooted in the node r where the depth-first search was started, then after completing the depth-first search, for each $i \in [n_T] \setminus \{r\}$, the entry $\pi[i]$ is the parent of i and $\pi[r]$ is undefined. Furthermore, we adopt the traditional notation used, for instance by Cormen et al. [Cor+09], of coloring the nodes of the tree white, gray, and black. A node i is colored white, when it was not yet discovered by the depth-first search. As soon as it is discovered, the node i becomes gray. Once it is finished, i. e., once all its neighbors except $\pi[i]$ are black, the node i becomes black. Often, it will also be useful to fill in another array D_G, whose entries correspond to the vertices of the underlying graph. Assume that $(T, \mathcal{X})$ is a tree decomposition of a graph G on n vertices with $V(G) = [n]$. The array D_G is used to store the discovery time of the vertices in G when performing a depth-first search on T and traversing the corresponding clusters simultaneously. More precisely, assume that all vertices of G are white in the beginning and that, when a node i of T turns gray, also every white vertex in its cluster turns gray. Then, $D_G[v]$ is the time when the vertex v of the graph G turns gray. Furthermore, let N and N' be integer arrays of length n_T each, where $N[i]$ is the size of the set X^i for all $i \in V_T$ and $N'[i] = |X^i \cap X^{\pi[i]}|$ for all $i \in V_T \setminus \{r\}$, i. e., the number of vertices that the clusters associated with the nodes i and its parent $\pi(i)$ have in common, and $N'[r] = 0$. The next lemma says that all these arrays can be computed during one depth-first traversal of the tree T in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time.

Lemma 2.30.
For every tree decomposition $(T, \mathcal{X})$ of some graph G with $V(T) = [n_T]$ for some integer n_T and $V(G) = [n]$ for some integer n, the arrays D, π, D_G, N, and N' can be computed with one depth-first traversal of T in time $\mathcal{O}(\|(T, \mathcal{X})\|)$. When a node i becomes gray, the entries $D[i]$, $\pi[i]$, $N[i]$, and $N'[i]$ are set correctly. Furthermore, for all $i \in V(T)$ and all $v \in X^i$, when i becomes gray, $D_G[v]$ is set or has been set previously and satisfies $D_G[v] \leq D[i]$.

Proof. Let $(T, \mathcal{X})$, G, n_T, and n be as in the statement, and let $T = (V_T, E_T)$ as well $\mathcal{X} = (X^i)_{i \in V_T}$. Moreover, fix a node $r \in V_T$ and assume that the considered depth-first traversal of T starts at r. Clearly, the arrays D and π will be set correctly. Additionally to steps of the depth-first search, every time a node $i \in V_T$ turns gray, the algorithm traverses the cluster X^i to compute the value $N[i]$ and to determine the number ℓ_i of vertices, which have their value in D_G already defined. Furthermore, when $i \in V_T$ turns gray, for all vertices $v \in X^i$ with $D_G[v]$ undefined, the algorithm sets the entry of $D_G[v]$ to $D[i]$. Afterwards, $D_G[v] \leq D[i]$ holds for all $v \in X^i$. When r turns gray, none of the entries in D_G has been defined before and the algorithm sets $N'[r] = 0$. At any time, the subgraph of T that is induced by the gray nodes is connected and, when i turns gray, then $\pi[i]$ is the only neighbor of i in this subgraph. Thus, (T3') implies that, for every $i \in V(T) \setminus \{r\}$, every vertex $v \in X^i$ with $D_G[v] < D[i]$ is also in the cluster $X^{\pi[i]}$. Therefore, $\ell_i = |X^i \cap X^{\pi[i]}|$ and the algorithm sets $N'[i] = \ell_i$.

Let us now analyze the running time of this procedure. The depth-first traversal itself requires $\mathcal{O}(n_T)$ time and, for all $i \in V_T$, it takes $\mathcal{O}(|X^i| + 1)$ additional time when the node i turns gray. Hence, the entire procedure takes $\mathcal{O}(\|(T, \mathcal{X})\|)$ time. □

In an iterative procedure, the algorithm only receives a tree decomposition for the input graph. So when a subroutine is applied to a subgraph of the input graph, the algorithm needs to compute a tree decomposition of the subgraph. As mentioned above, computing a tree decomposition of smallest possible width from scratch for a general graph is NP-hard. However, with the tree decomposition of the input graph at hand, it is easy to find a tree decomposition of a subgraph, namely the induced tree decomposition, see also Proposition 2.14. Consider a graph G on n vertices with $V(G) = [n]$ and a tree decomposition $(T, \mathcal{X})$ of G, where $\mathcal{X} = (X^i)_{i \in V(T)}$. Fix an arbitrary subgraph $\tilde{G}$ of G. Recall that $(\tilde{T}, \tilde{\mathcal{X}})$ with $\tilde{T} = T$ and $\tilde{\mathcal{X}} = (\tilde{X}^i)_{i \in V(\tilde{T})}$ is the induced tree decomposition of $\tilde{G}$ if $\tilde{X}^i = X^i \cap V(\tilde{G})$ for all $i \in V(T)$. To compute the induced tree decomposition of $\tilde{G}$, one basically has to traverse each cluster of the tree decomposition $(T, \mathcal{X})$ and delete each vertex that does not belong to $\tilde{G}$. In some cases, using the induced tree decomposition may cause many empty clusters. Whenever it is possible to predict which clusters will be empty, the following, more general concept is useful. The tree decomposition $(\tilde{T}, \tilde{\mathcal{X}})$ with $\tilde{\mathcal{X}} = (\tilde{X}^i)_{i \in V(\tilde{T})}$ is called the *restriction of* $(T, \mathcal{X})$ *to* $\tilde{T}$ *and* $\tilde{G}$ if $V(\tilde{T}) \subseteq V(T)$ and $\tilde{X}^i = X^i \cap V(\tilde{G})$ for all $i \in V(\tilde{T})$. So, compared to the induced tree decomposition, the tree $\tilde{T}$ can be chosen arbitrarily, but a restriction will only be a tree decomposition if $\tilde{T}$ is chosen well. For example every vertex of $\tilde{G}$ needs to be in a cluster of a node in $V(\tilde{T})$ in the original tree decomposition in order to satisfy (T1) and $\tilde{T}$ cannot have many edges in different places than T as otherwise (T3) is not satisfied for $(\tilde{T}, \tilde{\mathcal{X}})$. Note that an induced tree decomposition is a special case of a restriction of a tree decomposition. Therefore, the next proposition about the computation of a restriction of a tree decomposition also applies to induced tree decompositions.

Proposition 2.31.
Let G be an arbitrary graph and let $(T, \mathcal{X})$ be an arbitrary tree decomposition of G. For every graph $\tilde{G} \subseteq G$ and for every tree $\tilde{T}$ with $V(\tilde{T}) \subseteq V(T)$, the following holds for the restriction $(\tilde{T}, \tilde{\mathcal{X}})$ of $(T, \mathcal{X})$ to $\tilde{T}$ and $\tilde{G}$:

a) The width and the size of $(\tilde{T}, \tilde{\mathcal{X}})$ are at most the width and the size of $(T, \mathcal{X})$, respectively.

b) If, for every vertex $v \in V(G)$, one can check whether v is in $V(\tilde{G})$ in constant time, then one can compute $(\tilde{T}, \tilde{\mathcal{X}})$ in $\mathcal{O}\left(|V(\tilde{T})| + \sum_{i \in V(\tilde{T})} |X_0^i|\right)$ time, when given $(T, \mathcal{X})$ and $\tilde{T}$ as input.

Proof. Let G, $(T, \mathcal{X})$, $\tilde{G}$ and $\tilde{T}$ be as stated in the proposition and denote by $(\tilde{T}, \tilde{\mathcal{X}})$ with $\tilde{\mathcal{X}} = (\tilde{X}^i)_{i \in V(\tilde{T})}$ the restriction of $(T, \mathcal{X})$ to $\tilde{T}$ and $\tilde{G}$.

a) Neither the number of nodes of $\tilde{T}$ exceeds the number of nodes of T nor the number of vertices in the cluster $\tilde{X}^i$ exceeds the number of vertices in the cluster X^i for every $i \in V(\tilde{T})$. Thus, the statement follows.

b) Assume that there is a function that, for a vertex $v \in V(G)$, determines whether $v \in V(\tilde{G})$ in constant time. The algorithm traverses $\tilde{T}$, for example with a depth-first search, and while doing so, for each node i in $\tilde{T}$, it does the following. The algorithm accesses the corresponding node of T, which has a pointer to the cluster X^i, and then it computes $\tilde{X}^i$ by going through X^i and keeping only the vertices in $\tilde{G}$. Finally, it sets the pointer of i in $\tilde{T}$ to point to the new cluster $\tilde{X}^i$. So, processing a node $i \in V(\tilde{T})$ takes time proportional to $|X^i| + 1$. Therefore, the entire procedure takes $\mathcal{O}\left(|V(\tilde{T})| + \sum_{i \in V(\tilde{T})} |X^i|\right) = \mathcal{O}(\|(T, \mathcal{X})\|)$ time. □

Note that in Part b) the procedure to check whether a vertex $v \in V(G)$ is in $V(\tilde{G})$ is essential, as it might take too long to traverse the graph $\tilde{G}$ with a depth-first search or something similar in order to obtain a list of the vertices in $V(\tilde{G})$. For example, assume that G is composed of a path on $\frac{1}{2}n + 1$ vertices and a clique on $\frac{1}{2}n$ vertices, which have exactly one vertex in common. Let $\tilde{G}$ be the clique. Then, traversing $\tilde{G}$ takes $\Omega(n^2)$ time due to the large amount of edges. However, G admits a tree decomposition $(T, \mathcal{X})$ of width $\frac{1}{2}n$ with two clusters: One cluster, say X^1, corresponds to the clique and the other one, say X^2 corresponds to the path. Let $\tilde{T}$ be the tree on one node, which is the node whose cluster is X^1. Clearly, the restriction of $(T, \mathcal{X})$ to $\tilde{T}$ and $\tilde{G}$ is a tree decomposition of $\tilde{G}$. Proposition 2.31b) guarantees that if, for every $v \in V(G)$, one can check whether $v \in V(\tilde{G})$ in constant time, then the restriction of $(T, \mathcal{X})$ to $\tilde{T}$ and $\tilde{G}$ can be computed in $\mathcal{O}(n)$ time.

Consider a tree decomposition $(T, \mathcal{X})$ of some graph G and fix a subgraph $\tilde{G}$. Usually, the induced tree decomposition of $\tilde{G}$ has many empty clusters when $\tilde{G}$ is much smaller than G and the width of $(T, \mathcal{X})$ is constant. Such empty clusters can slow down some of the algorithms described ahead. Furthermore, repeated clusters can slow down the algorithm. For example, let i be a node of T, then attaching several copies of i to i itself with the same cluster as the cluster of the original node i gives an unnecessarily blown up tree decomposition of G. Therefore, the notion of a nonredundant tree decomposition is introduced, see also [KT06]. A tree decomposition $(T, \mathcal{X})$ of a graph G with $\mathcal{X} = (X^i)_{i \in V(T)}$ is called *nonredundant* if $X^i \not\subseteq X^j$ and $X^j \not\subseteq X^i$ for every edge $\{i, j\} \in E(T)$. This prevents the tree decomposition from being unnecessarily large to some extend, as a nonredundant tree decomposition can neither contain empty clusters nor two identical clusters whose nodes are adjacent. Furthermore, for any two clusters X^i and X^j corresponding to two distinct nodes i and j of a nonredundant tree decomposition $(T, \mathcal{X})$, it follows that $X^i \not\subseteq X^j$ and $X^j \not\subseteq X^i$. Indeed, assume that $i \neq j$ and $X^i \subseteq X^j$. Then, (T3) implies that $X^i \subseteq X^h$ for every h on the unique i,j-path in T. So let i' be the node after i on the i,j-path in T. Then, the edge $\{i, i'\}$ shows that $(T, \mathcal{X})$ is not nonredundant. Requiring a tree decomposition to be nonredundant does not force it to have the least possible amount of nodes among all tree decompositions with a certain width of the same underlying graph. Nevertheless, for the algorithms presented here, it suffices to have an upper bound on the number of nodes in the decomposition tree, as presented in the next lemma. The lemma also states that a tree decomposition can be made nonredundant in linear time without increasing its width. A proof for Part a) of the following proposition can also be found in the literature, see Fact 10.16 in [KT06] or Fact 1.9 in [Ree97].

Proposition 2.32.
For every graph G and every tree decomposition $(T, \mathcal{X})$ of G of width $t - 1$, the following holds.

a) If $(T, \mathcal{X})$ is nonredundant, then $|V(T)| \leq |V(G)|$ and $\|(T, \mathcal{X})\| = \mathcal{O}(|V(G)|t)$.

b) By contracting some of the edges of T one can transform $(T, \mathcal{X})$ into a nonredundant tree decomposition $(T', \mathcal{X}')$ such that $\|(T', \mathcal{X}')\| \leq \|(T, \mathcal{X})\|$ and the width of $(T', \mathcal{X}')$ is $t - 1$. If the underlying graph G satisfies $V(G) = [n]$, then such a tree decomposition $(T', \mathcal{X}')$ can be computed in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time.

Before presenting the proof of the previous proposition, arborescences are introduced, which are useful for the algorithm contained in Part b). An *arborescence* is a rooted tree T, where each node $i \in V(T)$ has a list of its children and knows its parent. The next lemma says that it takes linear time to convert a tree given by its adjacency lists into an arborescence and vice versa.

Lemma 2.33.
For every tree T on n_T vertices, it takes $\mathcal{O}(n_T)$ time to convert T into an arborescence when T is given by its adjacency lists and vice versa. Furthermore, when given an arborescence with root r, it takes $\mathcal{O}(n_T)$ time to compute an arborescence of the same underlying tree with a different root.

Let us quickly explain the procedure contained in Lemma 2.33. Consider a tree T with $V(T) = [n_T]$ and fix a node r that is to be used as the root of T. First, the algorithm marks r to be the root by setting $p(r) = r$ and then, it traverses the tree T with a depth-first search starting at r. When a node $i \in V(T)$ turns gray, the algorithm traverses the adjacency list of i and, for each node j there, it sets $p(j) = i$, except in the case $j = p(i)$, when it deletes j from the adjacency list of i. Note that $p(i)$ has been determined correctly when i turns gray, and after completing the depth-first traversal the adjacency list of each node i contains exactly the children of i. Processing one node $i \in V(T)$ takes time proportional to $\deg_T(i)$ and, hence, the entire procedure to turn T into an arborescence takes $\mathcal{O}(|V(T)|)$ time. To convert an arborescence T with root r to a tree stored by its adjacency lists note that, for each node $i \neq r$, the algorithm simply needs to add $p(i)$ to the list of children of i to obtain the adjacency list of i and the adjacency list of r is simply the list of children of r. Clearly, this takes $\mathcal{O}(n_T)$ time.

Proof of Proposition 2.32. Let $G = (V, E)$ be an arbitrary graph on n vertices and let $(T, \mathcal{X})$ be a tree decomposition of G of width $t - 1$, with $T = (V_T, E_T)$ and $\mathcal{X} = (X^i)_{i \in V_T}$. Root T at an arbitrary node r and define $n_T := |V_T|$.

a) Suppose $(T, \mathcal{X})$ is nonredundant. To prove that $|V_T| \leq n$, label each node of T with a vertex of G in the following way. First, as $(T, \mathcal{X})$ is nonredundant, the cluster X^r is nonempty and $X^i \not\subseteq X^{p(i)}$ for every $i \neq r$. So, label the node r with an arbitrary vertex in X^r, and label each node $i \neq r$ with an arbitrary vertex in $X^i \setminus X^{p(i)}$. For a contradiction, suppose that two distinct nodes i and i' receive the same label $v \in V$. If necessary, exchange i and i' such that $p(i)$ is on the unique path between i and i'. Since $v \in X^i \cap X^{i'}$, (T3) implies that $v \in X^{p(i)}$, which contradicts the choice of v as label for i. As there are only n distinct labels, the tree T has at most n nodes. Moreover, $\|(T, \mathcal{X})\| = |V_T| + \sum_{i \in V_T} |X^i| \leq n_T + tn_T \leq (t+1)n$.

b) If T has only one node, then $(T, \mathcal{X})$ is already nonredundant. So, assume that $|V_T| \geq 2$. It is easy to see that $(T, \mathcal{X})$ can be modified to be nonredundant while still satisfying (T1)-(T3) by successively contracting each edge $\{i, j\}$ of T with $X^i \subseteq X^j$ to one node with cluster X^j. Let $(T', \mathcal{X}')$ be the resulting tree decomposition. As T' contains at most n_T nodes and $\mathcal{X}'$ is obtained from $\mathcal{X}$ by deleting clusters, the width and the size of $(T', \mathcal{X}')$ are at most the width and the size of $(T, \mathcal{X})$, respectively.

   Next, it is explained how to implement this procedure to run in the desired time under the assumption that $V(G) = [n]$ and $V_T = [n_T]$. First of all, the algorithm turns T into an arborescence with root r, which requires $\mathcal{O}(n_T)$ time by Lemma 2.33. While doing so, for each node $i \in V_T \setminus \{r\}$, the algorithm also stores a pointer to the entry i in the list of children of $p(i)$. This does not increase the asymptotic running time of turning T into an arborescence as the additional pointer of i can be set when $p(i)$ is set and, hence, the algorithm does not need to search the adjacency list of i. This will be convenient for contracting edges. For an edge $\{i, j\} \in E_T$, we say that $\{i, j\}$ is *contracted into* j, if the node resulting from the contraction is called j again and its cluster is X^j. The algorithm searches for the edges that need to be contracted and contracts them during the search. To do so, it traverses T with a depth-first search starting at r and computes the arrays D, D_G, N, and N' as in Lemma 2.30. Recall that the arrays D and D_G store the discovery times of the nodes of T and the vertices of G, respectively, and, for every $i \in V_T$, the values $N[i] = |X^i|$ and $N'[i] = |X^i \cap X^{p(i)}|$ are set correctly when i turns gray. Furthermore, when a node $i \neq r$ turns gray, the algorithm checks whether one of the following cases applies.

Case 1: $N[i] = N'[i]$. That means that $|X^i| = |X^i \cap X^{p(i)}|$ and, hence, $X^i \subseteq X^{p(i)}$. In this case, the algorithm contracts i into $p(i)$. To do so, the algorithm removes i from the list of children of $p(i)$. Furthermore, the algorithm appends the list of children of i to the list of children of $p(i)$ and for each node j in the list of children of i, it sets $p(j)$ to $p(i)$ and updates the additional pointer of j.

Case 2: $N[i] \neq N'[i]$ and $N'[i] = N[p(i)]$. That means that $|X^i| \neq |X^i \cap X^{p(i)}| = |X^{p(i)}|$, which implies $X^{p(i)} \subseteq X^i$ and $X^{p(i)} \neq X^i$. In this case, the algorithm contracts $p(i)$ into i. To do so, it contracts i into $p(i)$ and updates the pointer of $p(i)$ to its cluster such that it points to X^i. Moreover, it sets $N[p(i)] = N[i]$ and $D_G[v] = D[p(i)]$ for every $v \in X^i$ with undefined $D_G[v]$.

If one of the above cases applies and also when none of them applies, the algorithm continues the depth-first search on the possibly modified tree including the computation of the arrays.

Observe that neither the pair $(T, \mathcal{X})$ nor one of the arrays D, D_G, N, and N' is modified when a gray node turns black. The following invariants hold every time after a node i turned gray and the tree was modified according to Case 1 or Case 2, if applicable. Denote by $(T_c, \mathcal{X}_c)$ the state of the pair $(T, \mathcal{X})$ after these modifications.

(i) For each node j in T_c, the parent of j in T_c is stored in $p(j)$ and the additional pointer of j in the arborescence is set correctly.

(ii) For each node j in T_c that is gray, the entries $N[j]$ and $D[j]$ are set correctly with respect to a depth-first search in T_c.

(iii) For each vertex v in G, the following holds. If there is a gray or a black node j in T_c such that v is in the cluster of j in the tree decomposition $(T_c, \mathcal{X}_c)$, then $D_G[v] \leq D[j]$. Otherwise $D_G[v]$ is undefined.

(iv) The value $N'[i]$ is computed correctly with respect to $(T_c, \mathcal{X}_c)$.

(v) The pair $(T_c, \mathcal{X}_c)$ is a tree decomposition of G.

(vi) Each edge $\{j, j'\}$ in T_c where j is gray or black and j' is gray or black satisfies $X_c^j \not\subseteq X_c^{j'}$ and $X_c^{j'} \not\subseteq X_c^j$ where X_c^j and $X_c^{j'}$ denote the clusters of j and j' in the tree decomposition $(T_c, \mathcal{X}_c)$.

Clearly, (i)-(vi) are satisfied before r turns gray, which is the first node of the input tree that turns gray. Consider one step of the depth-first search when a node i turns gray and assume that (i)-(vi) are satisfied before i turns gray. It is easy to check that the parents and the additional pointers of the arborescence are modified correctly when the tree is modified. Additionally, observe that the arrays N, D, and D_G are updated correctly. Hence, (i)-(iii) are satisfied after i turned gray. As the computation of $N'[i]$ relies only on the array D_G, (iv) is satisfied as well when i turns gray. Observe that, if the tree decomposition is modified when i turns gray, then $X^i \subseteq X^{p(i)}$ or $X^{p(i)} \subseteq X^i$, and it is easy to check that (v) is satisfied.

To see that (vi) is satisfied after i turned gray, consider first an edge $\{j, j'\}$ in T_c that does not contain i or $p(i)$. Then, neither j nor j' changes its color or its cluster when i turns gray. Thus, it suffices to consider the edges of T_c that contain i or $p(i)$. As each child j of i is white when i turns gray, there is nothing to show for these edges $\{j, i\}$. Next, consider the edge $\{i, p(i)\}$ and denote by X^i and $X^{p(i)}$ the clusters of i and $p(i)$, respectively, before applying the modifications done when i turns gray. If $X^i = X^{p(i)}$ or $X^i \subseteq X^{p(i)}$, then $N[i] = N'[i]$ and the edge $\{i, p(i)\}$ is contracted according to Case 1. If $X^i \nsubseteq X^{p(i)}$ and $X^{p(i)} \subseteq X^i$, then $N[i] \neq N'[i]$ and $N'[i] = N[p(i)]$, which means that the edge $\{i, p(i)\}$ is contracted due to Case 2. Otherwise, neither $X^i \subseteq X^{p(i)}$ nor $X^{p(i)} \subseteq X^i$ and neither Case 1 nor Case 2 applies, which means that $\{i, p(i)\}$ is not contracted. If $p(i)$ is the root, there is nothing more to show for (vi). Otherwise, let j be the parent of $p(i)$. Observe that j and $p(i)$ are both gray when i turns gray. If the cluster of $p(i)$ is not exchanged when i turns gray, there is nothing to show. So assume that the cluster of $p(i)$ is exchanged when i turns gray, i. e., Case 2 applies, $X^{p(i)}$ is updated to X^i and $X^{p(i)} \subseteq X^i$. As (vi) was satisfied before i turned gray, it follows that $X^{p(i)} \nsubseteq X^j$ and $X^j \nsubseteq X^{p(i)}$. Therefore, $X^i \nsubseteq X^j$ and $X^j \nsubseteq X^i$. Indeed, if the former was not satisfied, then (T3) implies $X^i \subseteq X^{p(i)}$, which contradicts the assumption of Case 2. If the latter was not satisfied, then $X^j \subseteq X^{p(i)}$ due to (T3), which is a contradiction as well. This completes the proof of the invariants.

Now, (v) and (vi) imply that the described procedure indeed returns a nonredundant tree decomposition of G. So it only remains to discuss the running time. Consider an arbitrary node $i \in V_T$. If applicable, the contraction of the edge $\{i, p(i)\}$ takes time proportional to $\deg_T(i)$, as the additional pointer of i that was computed when turning T into an arborescence can be used to find the entry i in the list of children of $p(i)$ in constant time. Since the edge $\{i, p(i)\}$ is contracted when i turns gray, each original list of children is traversed at most once and each modified list of children is never traversed for a contraction. Thus, in total, all contractions take $\mathcal{O}(n_T)$ time together. Next, consider the traversals of the clusters in $\mathcal{X}$. Other than the traversals done by the algorithm contained in Lemma 2.30, each cluster is traversed at most once when its node turns gray and Case 2 applies. So, the time needed for these additional traversals of clusters is at most $\mathcal{O}(\|(T, \mathcal{X})\|)$. The depth-first search itself is a search of the final decomposition tree, plus constant time per removed node. Consequently, the overall running time is $\mathcal{O}(n_T + \|(T, \mathcal{X})\|) = \mathcal{O}(\|(T, \mathcal{X})\|)$. □

Chapter 3

Planar Graphs

This chapter focuses on bisections in planar graphs. First, in Section 3.1, a general approach to construct a bisection by using separators is introduced. This approach will yield the bounds on the minimum bisection width from Theorem 1.10 and Theorem 1.11, which were stated in Section 1.2.3 and are proved here, as well as a technical looking inequality that relates the minimum bisection width of a planar graph and its tree-width. The latter is then used in Section 3.2 to prove Theorem 1.4 and Corollary 1.5, which were stated in Section 1.2.2 and study the structure of bounded-degree planar graphs with large minimum bisection width. Besides, Section 3.2 introduces grid-homogeneous graphs and derives a lower bound for the minimum bisection width in grid-homogeneous graphs. Finally, in Section 3.3, which studies the algorithmic use of the concept of grid-homogeneous graphs, it is shown that some questions related to asking whether a graph is grid-homogeneous are NP-hard, as well as that some parameters, that are related to measuring how grid-homogeneous a graph is, can be approximated within a constant factor in polynomial time.

3.1 Using Separators to Construct Exact Cuts

The aim of this section is to present a general approach for constructing a bisection and then to derive a technical bound that relates the tree-width and the minimum bisection width of a planar graph. The approach for constructing a bisection is introduced in Section 3.1.1 and uses separators to decompose the considered graph into smaller pieces. In Section 3.1.2, separators promised by the Planar Separator Theorem are used to derive the bound in Theorem 1.10 as well as a linear-time algorithm computing a bisection within this bound. In Section 3.1.3, it is shown that, when given a tree decomposition of width $t-1$, then a separator in the underlying graph of size at most t can be computed in linear time. Using these separators, the bound in Theorem 1.11 is proved and a corresponding polynomial-time algorithm is derived. Afterward, both types of separators are combined in Section 3.1.4, which yields a technical bound for the minimum bisection width in planar graphs and is the key for proving Theorem 1.4 and Corollary 1.5 in the next section. A similar, though different looking, approach for constructing bisections is used in Chapter 7 of [Sch13], where the bounds presented in Theorem 1.10 and Theorem 1.11 are derived as well. This section extends the results obtained in Chapter 7 in [Sch13], as the algorithms presented here are new and the bound derived in Section 3.1.4 is also new. Furthermore, all results presented in this section do not only hold for bisections but any exact cut.

3.1.1 Constructing an Exact Cut by Successively Removing Separators

The first aim of this section is to present an algorithm that computes an m-cut in a graph G under the assumption that there is some method to compute separators in the graph G and all its subgraphs, that is, a method to find a small subset of the vertices whose removal decomposes the graph into several not so big components. More precisely, consider a graph $G = (V, E)$ on n vertices and fix a constant $0 < c < 1$. Then, a subset $S \subsetneq V$ is called a *c-separator* in G if every component of $G - S$ contains at most cn vertices. For technical reasons, the set $S = V$ is also considered to be a *c-separator* in G. Sometimes, when the specific value of c is not relevant, we refer to such a set simply as a *separator*. For example, the Planar Separator Theorem says that every planar graph on n vertices has a $\frac{2}{3}$-separator S with $|S| \leq \sqrt{8n}$, see Theorem 3.4 or [LT79]. The idea to construct an m-cut (B, W) in a graph G is the following. The algorithm starts with two empty sets B and W, and decomposes G into smaller parts by removing a separator. Then, it greedily puts as many parts as possible into the set B such that $|B| \leq m$ is still satisfied. All remaining parts, except the first one, that did not fit into B, are put in the set W. Then, the algorithm goes on recursively in the part that was neither put in the set B nor the set W. This is repeated until the remaining graph contains at most one vertex. Then, all vertices not yet assigned to the set B or W, which are the vertices in the remaining graph and all vertices that belong to some separator that was removed, are distributed to the sets B and W. This procedure is stated formally in Algorithm 3.1, where the set W is kept implicit. The next lemma presents some invariants of Algorithm 3.1, before the width of the cut produced by Algorithm 3.1 and its running time are analyzed.

Lemma 3.1.
Consider an application of Algorithm 3.1 to a graph $G_0 = (V_0, E_0)$ on n_0 vertices with size-parameter $m \in [n_0]$. The following invariants hold after each execution of the while loop, where $V = V(G)$, B, and S are the sets used in Algorithm 3.1:

(i) the sets V, B, and S are pairwise disjoint,

(ii) $e_{G_0-S}(B, W, V) = 0$ for $W = V_0 \setminus (B \cup V \cup S)$, or equivalently, $G_0 - S$ decomposes into the disjoint parts $G[B]$, $G[W]$, and $G[V]$, as well as

(iii) $|B| \leq m \leq |B| + |S| + |V|$.

Proof. As in the statement, let $G_0 = (V_0, E_0)$ be a graph to which Algorithm 3.1 is applied and denote by m the size-parameter used in that application. Clearly, (i), (ii), and (iii) hold before the first execution of the while loop. Now, suppose that (i)-(iii) hold at the beginning of the s^{th} execution of the while loop for some $s \in \mathbb{N}$, i. e., they hold after the $(s-1)^{\text{st}}$ execution of the while loop if $s \geq 2$ or before the first execution if $s = 1$. We will prove that (i)-(iii) hold at the end of the s^{th} execution of the while loop. To do so, fix S, B, and G to the state of the corresponding variable before the s^{th} execution of the while loop and denote by S', B', and G' the state of the same variable after the s^{th} execution of the while loop, respectively. Let $\tilde{S}$ be the separator chosen in Line 3 and let k, $U_1, \dots, U_k$, and ℓ be as determined in Lines 4-5 during the s^{th} execution of the while loop. Furthermore, define $V := V(G)$ and $V' := V(G')$, $W := V_0 \setminus (B \cup S \cup V)$ and $W' := V_0 \setminus (B' \cup S' \cup V')$, as well as $\tilde{B} = B' \setminus B$ and $\tilde{W} = W' \setminus W$. Lines 3-4 imply that $\{\tilde{S}, U_1, \dots, U_k\}$ is a partition of V. Then, Lines 5-6 distribute some of these sets to B and S, and Line 7 sets G' to be either one of the components in the partition $\{\tilde{S}, U_1, \dots, U_k\}$ or the empty set. All remaining components form the set $\tilde{W}$ by construction, see Figure 3.1. Therefore, $(\tilde{S}, \tilde{B}, V', \tilde{W})$ is a cut in G and, by construction, cuts only edges incident to $\tilde{S}$, or equivalently

$$e_{G-\tilde{S}}(\tilde{B}, \tilde{W}, V') = 0. \tag{3.1}$$

Algorithm 3.1: Computes an m-cut based on a method to find a separator in the input graph and its subgraphs.

```
    Input: a graph $G$ on $n$ vertices and an integer $m \in [n]$.
    Output: an $m$-cut $(B, W)$ in $G$.
 1  $G_0 \leftarrow G,\ B \leftarrow \emptyset,\ S \leftarrow \emptyset,\ n \leftarrow |V(G)|$;
 2  While $n > 1$ do
 3      Find a separator $\tilde{S}$ in $G$;
 4      Let $k$ be the number of components of $G - \tilde{S}$ and if $k \geq 1$, let $(U_1, E_1), \dots, (U_k, E_k)$ be the
         components of $G - \tilde{S}$;
 5      If $k = 0$ then $\ell \leftarrow 0$ else Let $\ell \in [k]$ be the largest integer with $|B| + \sum_{h=1}^{\ell} |U_h| \leq m$;
 6      $B \leftarrow B \cup \left(\bigcup_{h \in [\ell]} U_h\right),\ S \leftarrow S \cup \tilde{S}$;
 7      If $\ell + 1 \leq k$ then $G \leftarrow (U_{\ell+1}, E_{\ell+1}),\ n \leftarrow |U_{\ell+1}|$ else $G \leftarrow (\emptyset, \emptyset),\ n \leftarrow 0$;
 8  Endw
 9  $S \leftarrow S \cup V(G),\ G \leftarrow (\emptyset, \emptyset)$;
10  Let $S_B \subseteq S$ be an arbitrary subset with $|S_B| = m - |B|$;
11  $B \leftarrow B \cup S_B$;
12  Return $(B, V(G_0) \setminus B)$;
```

This also implies that $\tilde{S}$, $\tilde{B}$, and V' are pairwise disjoint subsets of V. Using that $B' = B \cup \tilde{B}$, $S' = S \cup \tilde{S}$, and that (i) was satisfied before the s^{th} execution of the while loop, it follows that (i) is satisfied after the s^{th} execution. Furthermore, instead of removing all vertices in S' at once, removing the vertices in S from G_0 first and then the vertices in $\tilde{S}$ from G gives

$$e_{G_0 - S'}(B', W', V') \;\leq\; e_{G_0 - S'}(B, \tilde{B}, W, \tilde{W}, V') \;\leq\; e_{G_0 - S}(B, W, V) + e_{G - \tilde{S}}(\tilde{B}, \tilde{W}, V').$$

As (ii) is satisfied before the s^{th} execution of the while loop, (3.1) implies that $e_{G_0 - S'}(B', W', V') = 0$,

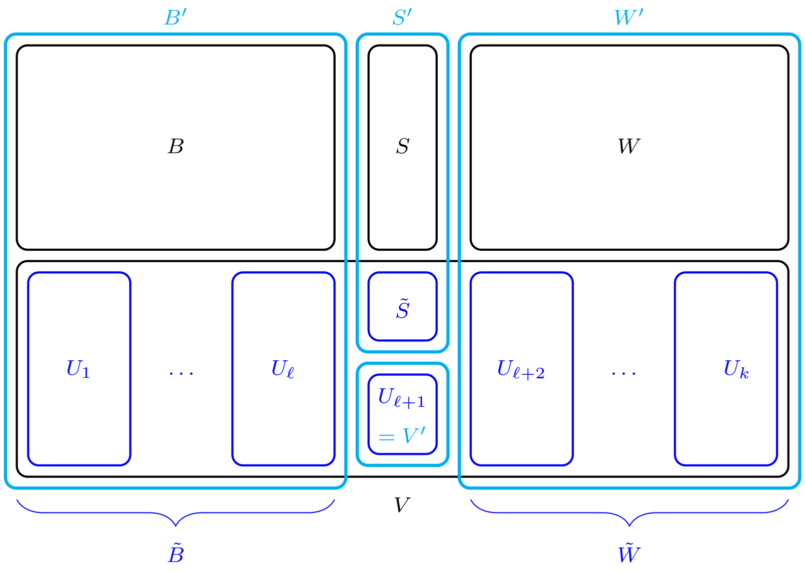


Figure 3.1: Notation used in the proof of Lemma 3.1.

i. e., (ii) is satisfied after the s^{th} execution of the while loop. To show that (iii) is satisfied, consider the following cases.

Case 1: $k = 0$. Then, $\tilde{S} = V$, $B' = B$, $S' = S \cup V$, and $V' = \emptyset$. Using that (iii) is satisfied before the s^{th} execution of the while loop, it follows that

$$|B'| \;=\; |B| \;\leq\; m \;\leq\; |B| + |S| + |V| \;=\; |B'| + |S'| + |V'|,$$

as V and S are disjoint by (i).

Case 2: $k \geq 1$. Then, the choice of ℓ in Line 5 implies that $|B'| \leq m$.

Case 2a: $\ell = k$. Then, $B' = B \cup (V \setminus \tilde{S})$, $S' = S \cup \tilde{S}$, and $V' = \emptyset$. Using that (iii) is satisfied before the s^{th} execution of the while loop, it follows that $m \leq |B| + |S| + |V| = |B'| + |S'| + |V'|$.

Case 2b: $\ell < k$. Then, $V' = U_{\ell+1}$. Furthermore, $B \cap U_h \subseteq B \cap V = \emptyset$ for all $h \in [k]$ as (i) is satisfied before the s^{th} execution of the while loop. Now, due to the choice of ℓ in Line 5, it follows that

$$|B'| + |S'| + |V'| \geq |B'| + |U_{\ell+1}| > m.$$

All in all, (iii) is satisfied after the s^{th} execution of the while loop. □

Lemma 3.2.
Consider an application of Algorithm 3.1 to a graph $G_0 = (V_0, E_0)$ on n_0 vertices with size-parameter $m \in [n_0]$. Algorithm 3.1 terminates and returns an m-cut (B, W) in G_0, that satisfies

$$e_{G_0}(B, W) \leq \Delta(G_0) \cdot |S|,$$

where S denotes the set of all removed vertices. More precisely, $S = \bigcup_{s=1}^{s^*} S_s$, where S_s denotes the separator $\tilde{S}$ used in the s^{th} execution of the while loop and s^* denotes the number of executions of the while loop.

Proof. Denote by $G_0 = (V_0, E_0)$ an arbitrary graph on n_0 vertices, fix an integer $m \in [n_0]$, and apply Algorithm 3.1. The number of vertices of the graph G in the algorithm, which is tracked by n, decreases by at least one in each execution of the while loop. Indeed, consider the separator $\tilde{S}$ used in Line 3. Then, each component of $G - \tilde{S}$ must have strictly less vertices than G, even if $\tilde{S}$ is empty, which might only happen in the first iteration if the input graph is not connected. Hence, there are at most n_0 executions of the while loop. Denote by s^* the number of executions of the while loop and, for $s \in [s^*]$, denote by S_s the separator $\tilde{S}$ used in Line 3 of the s^{th} execution of the while loop. As in the statement of the lemma, let $S = \bigcup_{s=1}^{s^*} S_s$. Invariants (i) and (iii) in Lemma 3.1 imply that $|B| \leq m \leq |B \,\dot{\cup}\, S|$ after Line 9 has been executed. Therefore, Line 10 is feasible and Algorithm 3.1 always returns an m-cut in the input graph. Furthermore, invariant (ii) in Lemma 3.1 implies that $e_{G_0 - S}(B, W, V) = 0$ holds for $W = V_0 \setminus (B \cup V \cup S)$ after the last execution of the while loop. If V is not empty at this point, then V contains only one vertex v and $e_{G_0 - S}(B \cup \{v\}, W) = 0$ as well as $e_{G_0 - S}(B, W \cup \{v\}) = 0$. Consequently, all cut edges of the returned m-cut must be incident to a vertex in S, which gives the desired bound on the width of the computed m-cut. □

Lemma 3.3.
Consider an application of Algorithm 3.1 to a graph $G_0 = (V_0, E_0)$ with $V_0 = [n_0]$ for some integer n_0 and with size-parameter $m \in [n_0]$. Furthermore, denote by $f_{\text{sep}}(G)$ the time needed to find a separator in the graph G in Line 3. Algorithm 3.1 computes an m-cut in G_0 in time proportional to

$$\sum_{s=1}^{s*} \left(f_{\text{sep}}(G_{s-1}) + \|G_{s-1}\| \right),$$

where G_s denotes the graph G in Algorithm 3.1 after the s^{th} execution of the while loop. In the implementation, bijections are set up, such that for the function $f_{\text{sep}}(G)$ one may assume that $V(G) = [n]$ for some integer n.

Proof. Consider a graph $G_0 = (V_0, E_0)$ with $V_0 = [n_0]$ for some integer n_0 and fix an arbitrary integer $m \in [n_0]$. Lemma 3.2 states that Algorithm 3.1 computes an m-cut in G_0 when applied to G_0 with size-parameter m. To implement Algorithm 3.1, all sets are stored as unordered lists and all graphs are stored by their adjacency lists. Furthermore, the algorithm keeps track of the size of each set and the size of the vertex set of each graph. Before executing Line 1 the algorithm sets up a bijection between $V(G_0)$ and $[n_0]$, for example the identity, and stores it in two arrays as in Lemma 2.21a), which takes $\mathcal{O}(n_0)$ time. Then, Line 1 takes $\mathcal{O}(\|G_0\|)$ time. Furthermore, Line 9 takes time proportional to $|V|$, as each vertex name needs to be converted back to its original name in G_0, which takes constant time for one vertex by Lemma 2.21b). As the set S_B can be constructed greedily, Lines 10-11 take $\mathcal{O}(m) = \mathcal{O}(n_0)$ time. So, except for the time consumed by the executions of the while loop, $\mathcal{O}(\|G_0\| + n_0) = \mathcal{O}(\|G_0\|)$ time is needed.

Now, consider one execution of the while loop and fix $G = (V, E)$ to be the state of the corresponding variable before this execution of the while loop. Assume that $V = [n]$ for $n := |V|$ and that there is a bijection that converts the integers in $[n]$ back to the original vertex names of the graph G_0. We need to argue that the considered execution of the while loop takes $\mathcal{O}(f_{\text{sep}}(G) + \|G\|)$ time including the time needed to adjust the bijection when the graph G is reset. Line 3 takes time proportional to $f_{\text{sep}}(G)$ and Line 4 takes $\mathcal{O}(n + |E|)$ by Corollary 2.24 and Lemma 2.25, including to determine lists of the vertices for each component (U_i, E_i) of $G - \tilde{S}$. Then, Line 5 takes time proportional to $\max\{k, 1\} \leq n$. In Line 6, the vertex names need to be converted back to the vertex names of the graph G_0, which takes constant time for each vertex by Lemma 2.21b). As in total at most n vertices are added to the set B and S together, Line 6 takes at most $\mathcal{O}(n)$ time. If $\ell + 1 > k$, then Line 7 takes constant time. Otherwise, let $n' := |U_{\ell+1}|$, which satisfies $n' < n$. Then, Line 7 takes $\mathcal{O}(n')$ time including the modification of the bijection by Corollary 2.23 and Lemma 2.21d), as $(U_{\ell+1}, E_{\ell+1}) = G[U_{\ell+1}]$ and a list of the vertices in $U_{\ell+1}$ has already been computed. All in all, the considered execution of the while loop takes $\mathcal{O}(f_{\text{sep}}(G) + \|G\|)$ time. Summing up gives the desired bound on the total running time. □

Before applying Algorithm 3.1 in Section 3.1.2-3.1.4, we briefly present a very easy application, which is discussed in detail in Chapter 7.2 in [Sch13], where also an example of an application of Algorithm 3.1 can be found. It is widely known and also not hard to show, that every tree T on n vertices contains a *separating vertex*, i. e., a vertex v such that each component of $T - v$ contains at most $\frac{1}{2}n$ vertices. Therefore, every tree T has a $\frac{1}{2}$-separator S with $|S| = 1$. When applying Algorithm 3.1 with these separators to a tree T_0 on n_0 vertices, then in each round of the while loop, the considered graph is a tree and its number of vertices decreases by at least a factor of $\frac{1}{2}$ in each round. So after s executions of the while loop, the considered tree has at most $\frac{1}{2^s}n_0$ vertices. Consequently, the while loop is executed at most $\log_2(n_0)$ times and the computed cut has width at most $\Delta(T_0)\log_2(n_0)$ by Lemma 3.2.

3.1.2 Using the Planar Separator Theorem

The aim of this section is to construct exact cuts in planar graphs by using Algorithm 3.1 with the separators promised by the Planar Separator Theorem of Lipton and Tarjan [LT79].

Theorem 3.4 (Planar Separator Theorem [LT79]).
Let $\sigma = \sqrt{8}$. Every planar graph G on n vertices has a $\frac{2}{3}$-separator S satisfying $|S| \le \sigma\sqrt{n}$. A separator S with these properties can be computed in $\mathcal{O}(n)$ time.

Note that, in [DV97], Djidjev and Venkatesan improved the size of a $\frac{2}{3}$-separator in a planar graph on n vertices to

$$\left(\sqrt{\tfrac{4}{3}} + \sqrt{\tfrac{2}{3}}\right)\sqrt{n} + \mathcal{O}(1) \approx 1.97\sqrt{n} + \mathcal{O}(1),$$

compared to $\sqrt{8n} \approx 2.83\sqrt{n}$. Furthermore, Djidjev [Dji82] showed that the smallest value for such a constant σ in the previous theorem must be at least $\frac{1}{3}\sqrt{4\pi\sqrt{3}} \approx 1.56$. In the following, we will use the separator theorem of Lipton and Tarjan, as the derived bounds on the cut width are only tight up to a constant factor, even when improved versions of the Planar Separator Theorem are used. As long as the separator can be computed in linear time, the bound on the running time remains valid as well.

Theorem 3.5 (Generalized version of Theorem 1.10).
Let $\sigma = \sqrt{8}$ and fix two arbitrary integers m and n with $1 \le m \le n$. Every planar graph G on n vertices has an m-cut (B, W) satisfying

$$e_G(B,W) \;\le\; c_\sigma \cdot \Delta(G) \cdot \sqrt{n},$$

where $c_\sigma = \sigma\left(3+\sqrt{6}\right)$. If $V(G) = [n]$, an m-cut satisfying this bound can be computed in $\mathcal{O}(n)$ time.

Proof. Fix $\sigma = \sqrt{8}$ and two integers m_0 and n_0 with $1 \le m_0 \le n_0$. Furthermore, let G_0 be an arbitrary planar graph on n_0 vertices. To derive the desired result, Algorithm 3.1 is applied to G_0 with size-parameter m_0 and using the $\frac{2}{3}$-separators promised by Theorem 3.4 in Line 3. This is feasible, because every graph G considered in Line 3 is a subgraph of the original graph and therefore planar. Let s^* be the total number of executions of the while loop, which is finite, as Algorithm 3.1 terminates by Lemma 3.2. For $s \in [s^*]$, denote by G_s and n_s the graph G and its number of vertices, respectively, after the sth execution of the while loop. Furthermore, for $s \in [s^*]$, let S_s be the separator in G_{s-1}, that is computed in Line 3 during the sth execution of the while loop. Then, S_s is a $\frac{2}{3}$-separator in G_{s-1} and $n_s \le \frac{2}{3}n_{s-1}$ for every $s \in [s^*]$, as the graph G_s is one of the components of $G_{s-1} - S_s$ or empty due to Line 7. Consequently,

$$n_s \;\le\; \left(\tfrac{2}{3}\right)^s n_0 \qquad \text{for every } s \in [s^*] \cup \{0\} \tag{3.2}$$

and

$$|S_s| \;\le\; \sigma\sqrt{n_{s-1}} \qquad \text{for every } s \in [s^*]$$

by Theorem 3.4. Defining $S := \bigcup_{s=1}^{s^*} S_s$ as in Lemma 3.2, the previous two equations imply that

$$\begin{aligned} |S| \;&\le\; \sum_{s=1}^{s^*} |S_s| \;\le\; \sigma \sum_{s=1}^{s^*} \sqrt{n_{s-1}} \;\le\; \sigma\sqrt{n_0} \sum_{s=1}^{s^*} \sqrt{\left(\tfrac{2}{3}\right)^{s-1}} \\ &\le\; \sigma\sqrt{n_0} \sum_{s=0}^{\infty} \left(\sqrt{\tfrac{2}{3}}\right)^s \;=\; \sigma \cdot \frac{1}{1-\sqrt{\frac{2}{3}}}\sqrt{n_0} \;=\; \sigma\left(3+\sqrt{6}\right)\sqrt{n_0}. \end{aligned}$$

Consequently, the computed m_0-cut (B, W) in G_0 satisfies

$$e_{G_0}(B, W) \;\leq\; \sigma\left(3 + \sqrt{6}\right)\Delta(G_0)\sqrt{n_0}$$

by Lemma 3.2.

To analyze the running time, observe that every planar graph $G = (V, E)$ satisfies $|E| \leq 3|V|$ due to Corollary 2.9. Hence, $\|G_s\| \leq 4n_s$ for all $s \in [s^*] \cup \{0\}$. Lemma 3.3 and the algorithm contained in Theorem 3.4 to compute the separators imply that the running time is

$$\mathcal{O}\left(\sum_{s=1}^{s^*}(f_{\text{sep}}(G_{s-1}) + \|G_{s-1}\|)\right) \;\leq\; \mathcal{O}\left(\sum_{s=1}^{s^*} n_{s-1}\right) \;\leq\; \mathcal{O}\left(\sum_{s=0}^{s^*-1}\left(\tfrac{2}{3}\right)^s n_0\right) \;\leq\; \mathcal{O}(n_0),$$

where (3.2) was used to estimate n_s. □

Next, it is quickly argued that the bound on the cut width in Theorem 3.5 is tight up to a constant factor for m-cuts in graphs on n vertices with $m = \Theta(n)$. Fix a constant $0 < \beta < \frac{1}{2}$. Let G be the $k \times k$ grid, define $n := k^2$, and fix an integer m with $\beta n \leq m \leq (1 - \beta)n$. The aim is to derive a lower bound on the width of an m-cut in G. Without loss of generality, one may assume that $m \leq \frac{1}{2}n$ as otherwise $m' := n - m$ can be considered and the black and white sets can be switched. Let (B, W) be an arbitrary m-cut in G, then Lemma 2.5 says that $e_G(B, W) \geq k \cdot \min\left\{\frac{1}{2}, \sqrt{\beta}\right\}$. Furthermore, as $\Delta(G) = 4$, Theorem 3.5 says that G allows an m-cut of width at most $(24\sqrt{2} + 16\sqrt{3})k$.

A nice property of the planar separators used in the proof of the previous theorem is that the bound on their size shrinks with the size of the considered subgraph. On the other hand, the size of the first separator might be quite large, even if the graph allows a bisection of not so large width. See [Sch13] for an example of a bounded-degree planar graph on n vertices that allows a bisection of constant width but the algorithm in Theorem 3.5 can produce a bisection of much larger width, in fact of width $\Omega(\sqrt{n})$. It is easy to modify this example for m-cuts in planar graphs on n vertices with $m = \Theta(n)$.

There are also separator theorems for planar graphs concerning *edge-separators*. For example in [Dik+93] it is shown that every planar graph $G = (V, E)$ on n vertices has a $\frac{2}{3}$-edge-separator of size $\sqrt{8\Delta(G)n}$, that is a set $F \subseteq E$ with the property that each component of $G - F$ contains at most $\frac{2}{3}n$ vertices. There, it is also shown that every planar graph on n vertices admits a bisection of width $\mathcal{O}(\sqrt{\Delta(G)n})$. Note that the dependence on the maximum degree of the graph in the bound on the minimum bisection width in [Dik+93] is asymptotically better than in Theorem 3.5.

3.1.3 Using Tree Decompositions

The aim of this section is to apply Algorithm 3.1 to an arbitrary graph with a given tree decomposition. To do so, a method to compute separators in the graph and all its subgraphs is required. These separators will be constructed from the given tree decomposition as every tree decomposition contains a cluster that is a separator in the underlying graph. More precisely, consider a graph $G = (V, E)$ on n vertices and a tree decomposition $(T, \mathcal{X})$ with $\mathcal{X} = (X^i)_{i \in V(T)}$ of G. A cluster X^i with $i \in V(T)$ is called a *separating cluster* if X^i is a $\frac{1}{2}$-separator in G, i. e., each component of $G - X^i$ contains at most $\frac{1}{2}n$ vertices or $X^i = V(G)$. The following lemma says that a separating cluster can be computed in time linear in the size of the provided tree decomposition. It is a generalization of the widely known fact that every tree T' contains a separating vertex.

Lemma 3.6.
For every graph $G = (V, E)$ and every tree decomposition $(T, \mathcal{X})$ of G, there exists a separating cluster in $\mathcal{X}$. If $V(G) = [n]$ for some integer n, then a separating cluster can be computed in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time and requires only the tree decomposition $(T, \mathcal{X})$ and the number n as input, but not the underlying graph G.

Before proceeding with the proof of this lemma, a technical definition and a proposition are presented. These are separated from the proof of the lemma, as they will be used again in Chapter 4. Consider a graph $G = (V, E)$ and a tree decomposition $(T, \mathcal{X})$ of G with $T = (V_T, E_T)$ and $\mathcal{X} = (X^i)_{i \in V_T}$. Assume that T is rooted at an arbitrary node r and recall that, for every node $i \in V_T \setminus \{r\}$, the parent of i in T is denoted by $p(i)$, as well as that each node i in T is considered to be a descendant of itself. Define

$$\begin{aligned}
Y^i &= \bigcup_{j \text{ descendant of } i} X^j && \text{for every } i \in V_T,\\
\tilde{Y}^i &= Y^i \setminus X^{p(i)} && \text{for every } i \in V_T \setminus \{r\},\\
\tilde{Y}^r &= Y^r,
\end{aligned}$$

as well as $y_i = |Y^i|$ and $\tilde{y}_i = |\tilde{Y}^i|$ for every $i \in V_T$.

Proposition 3.7.
For every graph G and every tree decomposition $(T, \mathcal{X})$ of G, where $T = (V_T, E_T)$ is a rooted tree and $\mathcal{X} = (X^i)_{i \in V_T}$, the following holds for the sets Y^i and $\tilde{Y}^i$ as defined above.

a) For every node $i \in V_T$,

$$Y^i = X^i \,\dot\cup \left(\dot{\bigcup_{j \text{ child of } i}} \tilde{Y}^j \right).$$

b) For every node $i \in V_T$ with children $j_1, j_2, \ldots, j_k$ and every partition $X^i = Z_1 \,\dot\cup\, Z_2$,

$$E_G(Z_1, Z_2, \tilde{Y}^{j_1}, \tilde{Y}^{j_2}, \ldots, \tilde{Y}^{j_k}, V(G) \setminus Y^i) \;\subseteq\; E_G(i).$$

c) If $V(G) = [n]$, then one can compute y_i and $\tilde{y}_i$ for every $i \in V_T$, all together in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time.

Proof. Let $G = (V, E)$ be an arbitrary graph and let $(T, \mathcal{X})$ be an arbitrary tree decomposition of G with $T = (V_T, E_T)$ and $\mathcal{X} = (X^i)_{i \in V_T}$. Assume that T is a rooted tree and denote by r its root.

a) Fix an arbitrary node $i \in V_T$. By definition,

$$Y^i \;=\; X^i \cup \left(\bigcup_{j \text{ child of } i} Y^j \right) \;=\; X^i \cup \left(\bigcup_{j \text{ child of } i} \tilde{Y}^j \right).$$

Clearly, X^i and $\tilde{Y}^j$ are disjoint for every child j of i. So it remains to show that the sets $\tilde{Y}^j$ are pairwise disjoint for all children j of i. Let k be the number of children of i and assume that $k \geq 2$ as otherwise there is nothing left to prove. Denote by $j_1, j_2, \ldots, j_k$ the children of i in T. For every $\ell \in [k]$, let V_ℓ^T be the node set of the component of $T - i$ that contains j_ℓ. Lemma 2.16a) implies that removing the vertices in X^i decomposes G into pairwise disjoint parts, among which the disjoint parts $G[V_1], G[V_2], \ldots, G[V_k]$, where

$$V_\ell \;=\; \bigcup_{h \in V_\ell^T} \left(X^h \setminus X^i\right) \;=\; \tilde{Y}^{j_\ell} \qquad \text{for each } \ell \in [k].$$

The last equality holds because V_ℓ^T is the node set of the subtree of T which is rooted in j_ℓ. Therefore, the sets $\tilde{Y}^{j_1}, \tilde{Y}^{j_2}, \ldots, \tilde{Y}^{j_k}$ are pairwise disjoint by Lemma 2.16a).

b) Fix some $i \in V_T$. If i is a leaf in T, then the statement is clear, so assume that i has at least one child. Denote by $j_1, \dots, j_k$ the children of i and consider an arbitrary partition $Z_1 \dot\cup Z_2$ of X^i. In the proof of Part a) it was shown that $G - X^i$ decomposes into disjoint parts among which $\tilde{Y}^{j_1}, \tilde{Y}^{j_2}, \dots, \tilde{Y}^{j_k}$. All other vertices of $G - X^i$ are in $V \setminus Y^i$ as $X^i \cup \bigcup_{\ell=1}^{k} \tilde{Y}^{j_\ell} = Y^i$ due to Part a). So, $G - X^i$ decomposes into the disjoint parts $\tilde{Y}^{j_1}, \tilde{Y}^{j_2}, \dots, \tilde{Y}^{j_k}$ and $V \setminus Y^i$. As all vertices in X^i are isolated in $G - E_G(i)$,

$$E_{G-E_G(i)}(Z_1, Z_2, \tilde{Y}^{j_1}, \tilde{Y}^{j_2}, \dots, \tilde{Y}^{j_k}, V \setminus Y^i) = \emptyset,$$

which is equivalent to the statement.

c) Assume that $V(G) = [n]$. To compute the y_i's and $\tilde{y}_i$'s, observe that

$$y_i = \sum_{j \text{ child of } i} \tilde{y}_j + |X^i| \qquad \text{for every } i \in V_T$$

by Part a), and $\tilde{y}_i = y_i - |X^i \cap X^{p(i)}|$ for every $i \in V_T \setminus \{r\}$, since every vertex in $X^{p(i)} \cap Y^i$ must be in X^i due to property (T3') of tree decompositions. The algorithm traverses the tree T and the clusters in $\mathcal{X}$ with a depth-first search as in Lemma 2.30, where among others, two arrays N and N' of length $n_T := |V_T|$ are filled in. For each $i \in V_T$, the entry $N[i]$ contains $|X^i|$ and the entry $N'[i]$ is set to $|X^i \cap X^{p(i)}|$ when the node i turns gray. The values y_i and $\tilde{y}_i$ can be stored in two additional arrays of length n_T. Whenever a node $i \in V_T$ turns black, the algorithm computes y_i and $\tilde{y}_i$ with the above formulas. This is feasible as all children of i are black when i turns black and the values $|X^i|$ and $|X^i \cap X^{p(i)}|$ were already computed when i turned gray.

The traversal as in Lemma 2.30 takes $\mathcal{O}(\|(T, \mathcal{X})\|)$ time. Initializing the additional arrays takes $\mathcal{O}(n_T)$ time and at each node $i \in V_T$ the algorithm spends $\mathcal{O}(\deg_T(i))$ additional time to compute y_i and $\tilde{y}_i$. Hence, the additional time sums up to

$$\mathcal{O}(n_T) + \sum_{i \in V_T} \mathcal{O}(\deg_T(i)) \;=\; \mathcal{O}(n_T + |E_T|) \;=\; \mathcal{O}(n_T) \;=\; \mathcal{O}(\|(T, \mathcal{X})\|).$$

□

Proof of Lemma 3.6. Let $G = (V, E)$ be an arbitrary graph on n vertices and let $(T, \mathcal{X})$ be a tree decomposition of G with $T = (V_T, E_T)$ and $\mathcal{X} = (X^j)_{j \in V_T}$. If T contains only one node or $n = 1$, then there is a cluster X^i in $\mathcal{X}$ that contains all vertices of G and, hence, X^i is a separating cluster. So, in what follows, assume that T has at least two nodes and $n > 1$. Moreover, assume that T is rooted in some node r and define Y^i, $\tilde{Y}^i$, y_i, and $\tilde{y}_i$ for every $i \in V_T$ as done before Proposition 3.7. Let i^* be a node in T with

$$y_{i^*} > \tfrac{1}{2}n \qquad \text{and} \qquad y_j \le \tfrac{1}{2}n \quad \text{for all children } j \text{ of } i^*. \tag{3.3}$$

We claim that such a node i^* always exists and that X^{i^*} is a separating cluster. The algorithm contained in Lemma 3.6 then follows immediately from Proposition 3.7c). Observe that the algorithm in Proposition 3.7c) relies on a depth-first-search and computes the value y_i when the node i turns black. So the algorithm can stop as soon as it has found the first node i with $y_i > \frac{1}{2}n$. As the children of i are already black when i turns black and the algorithm did not stop before i, each child j of i satisfies $y_j \le \frac{1}{2}n$.

Note that $y_r = n$ as $Y^r = V$, and $y_{p(i)} \ge y_i$ for all $i \in V_T \setminus \{r\}$ as $Y^i \subseteq Y^{p(i)}$. So if r does not satisfy (3.3), then r has a child r' with $y_{r'} > \frac{1}{2}n$. Again, if r' does not satisfy (3.3), then r' has a child r'' with $y_{r''} > \frac{1}{2}n$. This produces a finite sequence of nodes that either ends with a node satisfying (3.3) or with a leaf i of T with $y_i > \frac{1}{2}n$, which also satisfies (3.3). Hence, there is always a node $i^* \in V_T$ with the property in (3.3).

Consider a node i^* with the property in (3.3). If i^* has no children, then $|X^{i^*}| = |Y^{i^*}| > \frac{1}{2}n$ and X^{i^*} is clearly a separating cluster. So assume that i^* has at least one child and let $j_1, j_2, \dots, j_k$ be the children of i^*. Proposition 3.7b) implies that $G - E_G(i^*)$ decomposes into disjoint parts with the vertex sets $X^{i^*}, \tilde{Y}^{j_1}, \tilde{Y}^{j_2}, \dots, \tilde{Y}^{j_k}, V \setminus Y^{i^*}$. Therefore, $G - X^{i^*}$ decomposes into disjoint parts with vertex sets $\tilde{Y}^{j_1}, \tilde{Y}^{j_2}, \dots, \tilde{Y}^{j_k}, V \setminus Y^{i^*}$, whose sizes are $\tilde{y}_{j_1}, \tilde{y}_{j_2}, \dots, \tilde{y}_{j_k}, n - \tilde{y}_{i^*}$, respectively. As i^* satisfies (3.3), each of these parts has size at most $\frac{1}{2}n$. Consequently, each component of $G - X^{i^*}$ contains at most $\frac{1}{2}n$ vertices and X^{i^*} is a separating cluster. □

Proposition 2.32a) implies that if the tree decomposition given to the algorithm contained in Lemma 3.6 is nonredundant and has width $t-1$, then the algorithm takes $\mathcal{O}(nt)$ time. In particular, if the graph is planar and a tree decomposition of minimum width is given, the running time is $\mathcal{O}(n^{\frac{3}{2}})$ by Proposition 2.13. The next theorem relies on Algorithm 3.1 applied with the separators from Lemma 3.6.

Theorem 3.8 (generalized version of Theorem 1.11).
For every graph G on n vertices, every integer $m \in [n]$, and every tree decomposition $(T, \mathcal{X})$ of G of width $t-1$, there exists an m-cut (B, W) in G with

$$e_G(B, W) \;\le\; t \log_2(n) \cdot \Delta(G).$$

If $V(G) = [n]$, an m-cut with these properties can be computed in $\mathcal{O}\left((\|G\| + \|(T, \mathcal{X})\|) \log_2(n)\right)$ time. If the provided tree decomposition is nonredundant, such an m-cut can be computed in $\mathcal{O}(nt)$ time.

Proof. Consider an arbitrary graph $G_0 = (V_0, E_0)$ on n_0 vertices and a tree decomposition $(T_0, \mathcal{X}_0)$ of G_0 of width $t-1$. Fix an integer $m \in [n_0]$. Algorithm 3.1 is applied to construct an m-cut in G_0, which requires a method to find separators in G_0 and every subgraph $G \subseteq G_0$. To apply Lemma 3.6 to a subgraph $G \subseteq G_0$, a tree decomposition $(T, \mathcal{X})$ of G is needed. Indeed, the tree decomposition induced by G in $(T_0, \mathcal{X}_0)$ is a tree decomposition of width at most $t-1$, see Proposition 2.14. Therefore, Lemma 3.6 together with Proposition 2.14 provides a method to find $\frac{1}{2}$-separators of size at most t in G_0 and every subgraph of G_0. Next, the width of the m-cut in G_0 computed by Algorithm 3.1 with these separators is analyzed. Denote by s^* the total number of executions of the while loop, which is finite, as Algorithm 3.1 terminates by Lemma 3.2. For $s \in [s^*]$, let G_s and n_s be the graph G and its number of vertices, respectively, after the s^{th} execution of the while loop. Moreover, for $s \in [s^*]$, denote by S_s the separator in G_{s-1}, that is determined in Line 3 in Algorithm 3.1 during the s^{th} execution of the while loop. Then, S_s is a $\frac{1}{2}$-separator in G_{s-1} and $n_s \le \frac{1}{2}n_{s-1}$ for every $s \in [s^*]$, as the graph G_s is one of the components of $G_{s-1} - S_s$ or empty due to Line 7. Consequently,

$$n_s \;\le\; \left(\tfrac{1}{2}\right)^s n_0 \qquad \text{for every } s \in [s^*] \cup \{0\}, \tag{3.4}$$

which implies that $s^* \le \log_2(n_0)$. Furthermore, $|S_s| \le t$ for every $s \in [s^*]$ by Lemma 3.6 and Proposition 2.14. So, defining $S := \bigcup_{s=1}^{s^*} S_s$ as in Lemma 3.2 yields $|S| \le t \log_2(n_0)$. Consequently, the computed m-cut (B, W) in G_0 satisfies $e_{G_0}(B, W) \le t\Delta(G) \log_2(n_0)$ by Lemma 3.2.

To analyze the running time of Algorithm 3.1 with separating clusters, consider first the time needed to compute one separator in a subgraph G of G_0 in Line 3 with Lemma 3.6 and Proposition 2.14. A list of the vertices in G can be read off the bijection that maps the vertex set of G to $[n]$ and is available according to Lemma 3.3. Then, a tree decomposition $(T, \mathcal{X})$ of G with the properties in Proposition 2.14 can be computed in time proportional to $\|(T_0, \mathcal{X}_0)\|$ by Proposition 2.31b). Note that $\|(T, \mathcal{X})\| \le \|(T_0, \mathcal{X}_0)\|$, so the $\frac{1}{2}$-separator in G can be found in $\mathcal{O}(\|(T_0,\mathcal{X}_0)\|)$ time by applying the algorithm in Lemma 3.6. Then, $f_{\text{sep}}(G_{s-1}) = \mathcal{O}(\|(T_0,\mathcal{X}_0)\|)$ for all $s \in [s^*]$, where f_{sep} is defined as in Lemma 3.3. Moreover, $\|G_{s-1}\| \leq \|G_0\|$ for all $s \in [s^*]$ as G_s is a subgraph of G. Using the bound on s^* from above, Lemma 3.3 implies that the m-cut in G_0 can be computed in

$$\mathcal{O}\left(s^*(\|G_0\| + \|(T_0,\mathcal{X}_0)\|)\right) = \mathcal{O}\left((\|G_0\| + \|(T_0,\mathcal{X}_0)\|)\log_2(n_0)\right)$$

time, which gives the first bound on the running time.

To derive the second bound on the running time, assume that the tree decomposition $(T_0,\mathcal{X}_0)$ is nonredundant. The procedure from above is modified to ensure that all used tree decompositions are nonredundant. Fix an $\tilde{s} \in [s^*-1]$ and assume the tree decomposition $(T_{\tilde{s}-1},\mathcal{X}_{\tilde{s}-1})$ of $G_{\tilde{s}-1}$ has already been constructed. Instead of using $(T_0,\mathcal{X}_0)$ to construct a tree decomposition $(T_{\tilde{s}},\mathcal{X}_{\tilde{s}})$ of $G_{\tilde{s}}$, we now use $(T_{\tilde{s}-1},\mathcal{X}_{\tilde{s}-1})$ and apply the algorithm in Proposition 2.32b) to ensure that $(T_{\tilde{s}},\mathcal{X}_{\tilde{s}})$ is nonredundant. Then, computing $(T_{\tilde{s}},\mathcal{X}_{\tilde{s}})$ takes $\mathcal{O}(\|(T_{\tilde{s}-1},\mathcal{X}_{\tilde{s}-1})\|)$ time and computing a separating cluster for $G_{\tilde{s}}$ takes $\mathcal{O}(\|(T_{\tilde{s}},\mathcal{X}_{\tilde{s}})\|) = \mathcal{O}(\|(T_{\tilde{s}-1},\mathcal{X}_{\tilde{s}-1})\|)$ time. Proposition 2.32a) implies that $\|(T_s,\mathcal{X}_s)\| = \mathcal{O}(tn_s)$ for all $s \in [s^*-1] \cup \{0\}$ as the width of $(T_s,\mathcal{X}_s)$ is at most t. Defining f_{sep} as in Lemma 3.3, one obtains

$$f_{\text{sep}}(G_s) \;=\; \mathcal{O}(\|(T_{s-1},\mathcal{X}_{s-1})\|) \;=\; \mathcal{O}(tn_{s-1})$$

for all $s \in [s^*-1]$ and $f_{\text{sep}}(G_0) = \mathcal{O}(\|(T_0,\mathcal{X}_0)\|) = \mathcal{O}(tn_0)$. Furthermore, Proposition 2.17 shows that $\|G_s\| = n_s + |E(G_s)| \leq tn_s \leq tn_{s-1}$ for all $s \in [s^*-1]$ and $\|G_0\| \leq tn_0$. So, Lemma 3.3 together with (3.4) implies that the total running time is

$$\begin{aligned}\mathcal{O}\left(\sum_{s=1}^{s^*}\left(f_{\text{sep}}(G_{s-1}) + \|G_{s-1}\|\right)\right) \;&=\; \mathcal{O}\left(tn_0 + \sum_{s=1}^{s^*-1}\left(f_{\text{sep}}(G_s) + \|G_s\|\right)\right)\\ =\; \mathcal{O}\left(tn_0 + \sum_{s=1}^{s^*-1} tn_{s-1}\right) \;&=\; \mathcal{O}\left(tn_0 + t\sum_{s=1}^{s^*-1}\left(\tfrac{1}{2}\right)^{s-1} n_0\right) \;=\; \mathcal{O}(tn_0).\end{aligned}$$ □

Observe that there are values of m for which the bound on the width of an m-cut in Theorem 3.8 is tight up to a constant factor. Indeed, consider a perfect ternary tree G on n vertices and any tree decomposition $(T,\mathcal{X})$ of G of width 1. Then, any m-cut in G with $m = \left\lfloor \frac{1}{2}n \right\rfloor$ cuts $\Omega(\log_3(n)) = \Omega(\log_2(n))$ edges by Theorem 2.4 and Theorem 3.8 promises that there is an m-cut in G of width $4\log_2(n)$.

Furthermore, Theorem 3.8 contains two algorithms that compute an m-cut in a graph G with a given tree decomposition $(T,\mathcal{X})$. Let n be the number of vertices of G and let $t-1$ be the width of $(T,\mathcal{X})$. Denote by $\mathcal{A}_1$ the algorithm running in $B_1 = \mathcal{O}\left((\|G\| + \|(T,\mathcal{X})\|)\log_2(n)\right)$ time and denote by $\mathcal{A}_2$ the algorithm running in $B_2 = \mathcal{O}(nt)$ time. When not changing the implementations, both bounds B_1 and B_2 are asymptotically tight, and none of them dominates the other as n tends to infinity. Indeed, in the following, we present an example where B_1 is tight and B_2 is not, as well as another example where it is the other way around.

The idea to show that B_1 can be asymptotically tight is to construct a tree decomposition that is composed of two tree decompositions, where one of them represents only a small part of the graph but has roughly half the size of the entire tree decomposition. This is possible for square grids. Consider a $k \times k$ grid G'. Then, G' allows a nonredundant tree decomposition of width k with a decomposition tree on $\Theta(k^2)$ nodes with $\Theta(k^2)$ clusters of size k. Hence, its size is $\Theta(k^3)$, while the underlying graph has only k^2 vertices. See Section 2.3 for an example of a tree decomposition of a grid with these properties. The remaining part of the tree decomposition will be chosen in a way such that the grid is not split into pieces during the first executions of the while loop. Hence, the running time arises from often traversing

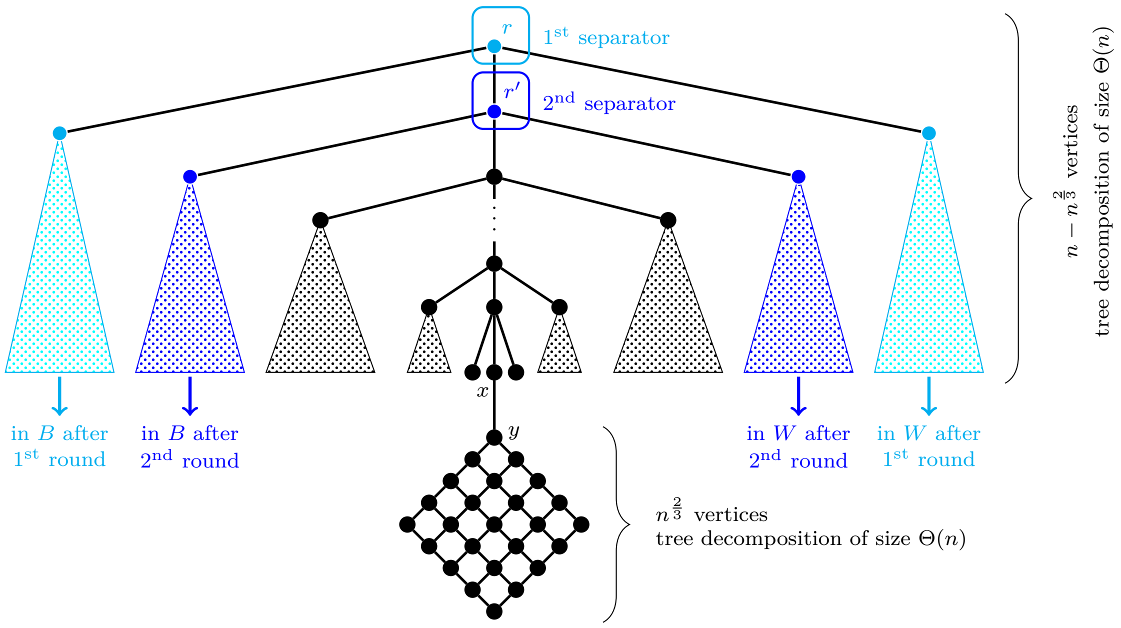

r
1st separator
r'
2nd separator
$n - n^{\frac{2}{3}}$ vertices
tree decomposition of size $\Theta(n)$
x
y
in B after 1st round
in B after 2nd round
in W after 2nd round
in W after 1st round
$n^{\frac{2}{3}}$ vertices
tree decomposition of size $\Theta(n)$

Figure 3.2: Example that shows that the bound $\mathcal{O}((\|G\| + \|(T, \mathcal{X})\|)\log_2(n))$ on the running time in Theorem 3.8 is tight.

the part of the tree decomposition that represents the grid. Let us now present the details of showing that B_1 can be asymptotically tight while B_2 is loose. Consider the following graph G on n vertices. Let $\tilde{G}_1$ be a perfect ternary tree on $n - n^{\frac{2}{3}}$ vertices with root r and let $\tilde{G}_2$ be a square grid on $n^{\frac{2}{3}}$ vertices.[1] Define G to be the graph obtained by inserting an edge between a leaf x of $\tilde{G}_1$ and an arbitrary vertex y of $\tilde{G}_2$, see Figure 3.2. Let $(\tilde{T}_1, \tilde{\mathcal{X}}_1)$ be a tree decomposition of $\tilde{G}_1$ of width 1, that is obtained from the usual tree decomposition of a tree by adding a cluster $X^{i_r} = \{r\}$ if it does not yet contain such a cluster, see Section 2.3 for an example of a tree decomposition of a tree with these properties. Note that $(\tilde{T}_1, \tilde{\mathcal{X}}_1)$ has size $\Theta(n)$. Moreover, let $(\tilde{T}_2, \tilde{\mathcal{X}}_2)$ be a tree decomposition of $\tilde{G}_2$ as the one described above, i. e., the width of $(\tilde{T}_2, \tilde{\mathcal{X}}_2)$ is $\Theta\left(\sqrt{n^{\frac{2}{3}}}\right) = \Theta\left(n^{\frac{1}{3}}\right)$ and the size of $(\tilde{T}_2, \tilde{\mathcal{X}}_2)$ is $\Theta(n)$. Assume the vertex sets of $\tilde{G}_1$ and $\tilde{G}_2$ are disjoint as well as that the node sets of $\tilde{T}_1$ and $\tilde{T}_2$ are disjoint. Join a node of $\tilde{T}_1$, whose cluster contains x, and a node of $\tilde{T}_2$, whose cluster contains y, by a path of length two to obtain a tree T. For the unique node i in T that is neither in $\tilde{T}_1$ nor in $\tilde{T}_2$ define $X^i = \{x, y\}$. Let $\mathcal{X}$ be the union of $\tilde{\mathcal{X}}_1$ and $\tilde{\mathcal{X}}_2$ plus the new cluster X^i. It is easy to see that $(T, \mathcal{X})$ is a tree decomposition of G of size $\Theta(n)$ and width $\Theta\left(n^{\frac{1}{3}}\right)$. When Algorithm $\mathcal{A}_1$ is applied to G with size-parameter $m = \left\lfloor \frac{1}{2}n \right\rfloor$, the first separating cluster can be X^{i_r}. Then, it is possible that the algorithm continues recursively in the component G' of $G - X^{i_r}$ that is composed of a perfect ternary tree of height one less than $\tilde{G}_1$ and the grid $\tilde{G}_2$. Denote by r' the neighbor of r that is on the unique r,x-path in $\tilde{G}_2$, let $i_{r'}$ be the node in $\tilde{T}_1$ with the cluster $X^{i_{r'}} = \{r, r'\}$ and let $(T', \mathcal{X}')$ be the tree decomposition induced by G' in $(T, \mathcal{X})$. Then, the cluster in $\mathcal{X}'$ that corresponds to $i_{r'}$ contains only the vertex r' and the same can happen as in the first round of the while loop of Algorithm 3.1. We refer to the phase of Algorithm $\mathcal{A}_1$ when this keeps repeating as the first phase and assume that the first phase lasts as long as possible. The first phase lasts at least as long as the remaining part of $\tilde{G}_1$ is at least three times as large as $\tilde{G}_2$, i. e., the remaining part

[1]Clearly, $n^{\frac{2}{3}}$ is not always a square number. If not, define $k := \left\lfloor n^{\frac{1}{3}} \right\rfloor$ and let $\tilde{G}_2$ be a $k \times k$ grid with a path of length $n^{\frac{2}{3}} - k^2$ attached to it. The remaining argument is easy to adjust.

of $\tilde{G}_1$ contains at least $3n^{\frac{2}{3}}$ vertices, as then the current root of the remaining part of $\tilde{G}_1$ is a $\frac{1}{2}$-separator in the remaining graph. Since the size of the remaining part of $\tilde{G}_1$ decreases roughly by a factor of $\frac{1}{3}$ in each round, the first phase lasts for $\Omega(\log n)$ rounds. In each round, a separating cluster in the remaining graph is computed and it can happen that, when doing so, the entire tree decomposition $(\tilde{T}_2, \tilde{\mathcal{X}}_2)$ is traversed, which takes $\Theta(n)$ time. This results in a total running time of $\Omega(n \log n)$, which matches the bound B_1, that evaluates to $\mathcal{O}(n \log n)$ using that the graph G contains $\mathcal{O}(n)$ edges by Corollary 2.9. Observe that when Proposition 2.32b) is applied in each round to make the tree decomposition of the current graph nonredundant and the next tree decomposition is computed from the one of the previous graph, then the running time does not decrease asymptotically. Indeed, then each separating cluster used in the first phase will contain two vertices, but it can happen that these two vertices are one vertex on the r,x-path in $\tilde{G}_1$ and one of its neighbors not on the r,x-path. In this example, the bound B_2 evaluates only to $\mathcal{O}\left(n \cdot n^{\frac{1}{3}}\right)$, which is asymptotically larger.

To see that B_2 can be asymptotically tight, while B_1 is loose, let G be a tree on n vertices and let $(T, \mathcal{X})$ be a tree decomposition of G of width one. Clearly, applying algorithm $\mathcal{A}_2$ to G takes $\Omega(n)$ time as a linear part of the tree decomposition needs to be searched in order to find the first separating cluster. This matches the bound B_2, which evaluates to $\mathcal{O}(n)$ in this case. The bound B_1 evaluates only to $\mathcal{O}(n \log n)$ in this case.

Observe that neither bound B_1 nor bound B_2 is linear in the size of the input; see also the discussion after Corollary 1.14 in Chapter 1. So one can ask whether it is possible to speed up one of the corresponding algorithms. As algorithm $\mathcal{A}_2$ traverses the entire tree decomposition to make it nonredundant in each execution of the while loop, this does not seem to be the case for $\mathcal{A}_2$. Concerning $\mathcal{A}_1$, the situation is better. Indeed, fix an input graph G_0 on n_0 vertices, let $m \in [n_0]$ be an arbitrary integer, and consider one execution of the while loop, where the graph G is broken into parts by removing a separating cluster arising from some tree decomposition $(T, \mathcal{X})$ of G. Recall the procedure of computing a separating cluster in $(T, \mathcal{X})$. To do so, the tree T was rooted in an arbitrary node r and some values y_i and $\tilde{y}_i$ were computed for all $i \in V(T)$, which is the major part of the running time of the algorithm, that computes a separating cluster. Assume that the cluster of some node i was chosen as separator and let j be a child of i. If $\tilde{Y}^j$ as defined before Proposition 3.7 induces the component of $G - X^i$ to which the graph G is reset, then computing the next separating cluster can be simplified. Indeed, the values $\tilde{y}_h$ for all nodes h in the subtree rooted in j are already computed and do not change, because Proposition 3.7a) implies that $\tilde{Y}^j$ and $\tilde{Y}^{j'}$ are disjoint for every child $j' \neq j$ of i. So it is at hand to work with the disjoint parts of $G - X^i$ defined in Lemma 2.16a) instead of the components of $G - X^i$. In fact, the proof of Lemma 3.6 shows that every tree decomposition has a node i' such that each disjoint part according to Lemma 2.16a) contains at most $\frac{1}{2}n'$ vertices, where n' denotes the number of vertices of the underlying graph. So from now on, assume that $\mathcal{A}_1$ works with the disjoint parts as in Lemma 2.16a) instead of components in Line 4 in Algorithm 3.1. Furthermore, we would like to ensure that the graph G is not reset to the part $G[V \setminus Y^i]$ after the separator $\tilde{S} = X^i$ was removed. Let B be the set of vertices assigned to the black set so far and let W be the white set, which is kept implicit, i. e., W is the set of all vertices that are neither in B, nor in one of the separators used so far, nor in the current graph G. Let n be the number of vertices of G. As B, W, and $V(G)$ are mutually disjoint, $\frac{1}{2}n$ vertices must fit into the set B or W without exceeding its size, i. e., $|B| + \frac{1}{2}n \leq m$ or $|W| + \frac{1}{2}n \leq n_0 - m$. Hence, the algorithm can first assign the part $G[V \setminus Y^i]$ to one of the sets and then distribute the remaining parts, such that it ensures that the graph G is reset to a part $G[\tilde{Y}^j]$ for a child j of i. However, in the next round, when the cluster of a node i' is chosen as separator, the size of the set $V' \setminus Y^{i'}$ cannot be estimated, where V' is the vertex set of the next graph G, as the values y_h might have changed and cannot be adjusted without traversing the clusters.

This difficulty can be solved by constructing the set B only from sets $\tilde{Y}^h$ for certain nodes $h \in V(T_0)$, except for the last step, where vertices from a set $X^{p(h)}$ might be used. This results in the strategy used in Chapter 4, where approximate cuts are constructed. There, the clusters that are implicitly removed to break the underlying graph into pieces are not necessarily separating clusters anymore, as the algorithm searches for nodes i such that the set Y^i contains enough vertices to fill up the current set B and for each child j of i the set $\tilde{Y}^j$ fits into the set B without exceeding m. In Chapter 4, Corollary 4.10 for exact m-cuts is derived, which contains an algorithm that computes a cut, whose width satisfies the same bound as the cut in Theorem 3.8, in linear time when given a tree decomposition as input.

To get back to Algorithm 3.1, let us quickly summarize what happens when it is used with separating clusters of a given tree decomposition. A nice property of the separating clusters is that the size of the separators is related to the tree-width of the considered graph, so the first separator is not so large if the tree-width of the graph is not so large. On the other hand, however, the size of the separator does not necessarily shrink during the application of Algorithm 3.1 when a subgraph of the previous graph is considered.

3.1.4 Using Planar Separators and Tree Decompositions

Consider a planar G on n vertices and a tree decomposition $(T, \mathcal{X})$ of G. Fix an integer $m \in [n]$ and denote by $t-1$ the width of $(T, \mathcal{X})$. When using Algorithm 3.1 to construct an m-cut (B, W) in G, the bound on the width of (B, W) is directly related to the size of the separators that were removed to break G and its considered subgraphs into smaller pieces, see Lemma 3.2. On the one hand, it is possible to use *planar separators*, i. e., separators promised by Theorem 3.4, as in Section 3.1.2. There, it was mentioned that the size of planar separators decreases as the size of the considered graph decreases, assuming that each separator is roughly as large as the bound provided by Theorem 3.4. However, in the first execution of the while loop of Algorithm 3.1, when a separator S in G has to be found, the size of S can be as large as $\Theta(\sqrt{n})$. Or in short, planar separators can be bad in the beginning but will be good later. On the other hand, as we assume to have a tree decomposition of G, it is possible to use separating clusters arising from $(T, \mathcal{X})$ as in Section 3.1.3. There, we saw that the size of a separating cluster does not necessarily decrease when the size of the considered graph decreases, since the width of the induced tree decomposition does not necessarily decrease as the underlying graph decreases. However, if t is not too large, the graph G allows a small separator S, which is good for the first execution of the while loop in Algorithm 3.1. Or in short, separating clusters are good in the beginning, but can be bad later. So, both types of separators have their advantages and disadvantages. Therefore, a natural idea is to consider both and use the smaller one. Analyzing the width of the corresponding m-cut results in the next theorem. It applies Algorithm 3.1 to G and the tree decomposition $(T, \mathcal{X})$, first with separating clusters arising from $(T, \mathcal{X})$ and later with planar separators. Its bound on the width of the constructed m-cut relates the results of Theorem 3.5 and Theorem 3.8. The bound on the width of the computed m-cut in G depends on t and a factor that is logarithmic in n, which will vanish for certain planar graphs.

Theorem 3.9.
Let $\sigma = \sqrt{8}$. For every planar graph G on n vertices, every integer $m \in [n]$, and every tree decomposition $(T, \mathcal{X})$ of G of width at most $t-1$ with $t \le \sigma\sqrt{n}$, there is an m-cut (B, W) in G that satisfies

$$e_G(B, W) \;\le\; t\left(\left\lceil \log_2\left(\frac{\sigma^2 n}{t^2}\right)\right\rceil + 3 + \sqrt{6}\right)\Delta(G).$$

If $V(G) = [n]$, an m-cut with these properties can be computed in $\mathcal{O}\left(\|(T, \mathcal{X})\| \cdot \log\left(\frac{2\sigma^2 n}{t^2}\right)\right)$ time. If the *tree decomposition $(T, \mathcal{X})$ is nonredundant, then an m-cut with these properties can be computed in $\mathcal{O}(nt)$ time.*

Proof. Fix $\sigma = \sqrt{8}$. Let $G_0 = (V_0, E_0)$ be an arbitrary planar graph on n_0 vertices and fix an integer $m \in [n_0]$. Furthermore, let $(T_0, \mathcal{X}_0)$ be a tree decomposition of G_0 of width at most $t-1$ with $t \leq \sigma\sqrt{n_0}$. Algorithm 3.1 is applied to G_0 with size-parameter m to construct an m-cut in G_0. All line numbers in the remaining proof refer to Algorithm 3.1. To apply Algorithm 3.1, a method to find separators in G_0 and all of its subgraphs is required for Line 3. Before specifying the separators, define the following two phases of the algorithm. Let

$$s_1' := \left\lceil \log_2\left(\frac{\sigma^2 n_0}{t^2}\right)\right\rceil, \tag{3.5}$$

which is non-negative due to the restriction $t \leq \sigma\sqrt{n_0}$. If there are more than s_1' executions of the while loop in Algorithm 3.1, then the first s_1' executions of the while loop are the first phase and the remaining ones are the second phase. If there are s_1' or less executions of the while loop, then the algorithm does not enter the second phase and the first phase refers to all executions of the while loop. During the first phase, separating clusters from the tree decomposition $(T_0, \mathcal{X}_0)$ and the tree decompositions induced by the current subgraphs of G in $(T_0, \mathcal{X}_0)$ are used in Line 3. Lemma 3.6 shows that this is feasible. In the second phase, planar separators, i. e., separators guaranteed by Theorem 3.4, are used in Line 3, which is feasible as all graphs G encountered during the second phase are subgraphs of the input graph G_0 and therefore planar.

Let s_2^* be the total number of executions of the while loop when Algorithm 3.1 is applied to G_0 in this way. Note that s_2^* is finite, as Algorithm 3.1 terminates by Lemma 3.2. Denote by s_1^* the end of the first phase, i. e., $s_1^* := \min\{s_1', s_2^*\}$. For $s \in [s_2^*]$, denote by G_s and n_s the graph G and its number of vertices, respectively, after the s^{th} execution of the while loop. For $s \in [s_1^* - 1]$, let $(T_s, \mathcal{X}_s)$ be the tree decomposition induced by G_s in $(T, \mathcal{X})$. Furthermore, for $s \in [s_2^*]$, let S_s be the separator in G_{s-1}, that is computed in Line 3 during the s^{th} execution of the while loop.

During the first phase, i. e., for $s \in [s_1^*]$, the separator S_s is a separating cluster in $(T_{s-1}, \mathcal{X}_{s-1})$ and therefore

$$|S_s| \;\leq\; t \qquad \text{for every } s \in [s_1^*], \tag{3.6}$$

as the width of $(T_{s-1}, \mathcal{X}_{s-1})$ is at most the width of $(T_0, \mathcal{X}_0)$. Moreover, S_s is a $\frac{1}{2}$-separator in G_{s-1} and $n_s \leq \frac{1}{2}n_{s-1}$ for every $s \in [s_1^*]$, as the graph G_s is one of the components of $G_{s-1} - S_s$ or empty due to Line 7. Consequently,

$$n_s \;\leq\; \left(\tfrac{1}{2}\right)^s n_0 \qquad \text{for every } s \in [s_1^*] \cup \{0\}$$

and in particular

$$n_{s_1^*} \;\leq\; \left(\tfrac{1}{2}\right)^{s_1^*} n_0 \;\leq\; \left(\tfrac{1}{2}\right)^{s_1'} n_0 \;\overset{(3.5)}{\leq}\; \left(\tfrac{1}{2}\right)^{\log_2\left(\frac{\sigma^2 n_0}{t^2}\right)} n_0 \;=\; \frac{t^2}{\sigma^2}\,. \tag{3.7}$$

During the second phase, i. e., for $s \in [s_2^*] \setminus [s_1^*]$, the separator S_s is a $\frac{2}{3}$-separator in G_{s-1}, as it is chosen according to Theorem 3.4. Since G_s is one of the components of $G_{s-1} - S_s$ or empty due to Line 7, it follows that $n_s \leq \frac{2}{3}n_{s-1}$ and also

$$n_s \;\leq\; \left(\tfrac{2}{3}\right)^{s-s_1^*} n_{s_1^*} \qquad \text{for every } s \in [s_2^*] \setminus [s_1^* - 1]. \tag{3.8}$$

Furthermore, Theorem 3.4 implies that

$$|S_s| \;\le\; \sigma\sqrt{n_{s-1}} \;\le\; \sigma\left(\sqrt{\tfrac{2}{3}}\right)^{s-s_1^*-1}\sqrt{n_{s_1^*}} \;\le\; t\left(\sqrt{\tfrac{2}{3}}\right)^{s-s_1^*-1} \qquad \text{for every } s \in [s_2^*]\setminus[s_1^*], \tag{3.9}$$

where (3.8) and (3.7) were used to estimate n_{s-1} and $n_{s_1^*}$, respectively.

Defining $S := \bigcup_{s=1}^{s_2^*} S_s$ as in Lemma 3.2, equations (3.6) and (3.9) imply that

$$\begin{aligned} |S| \;&\le\; \sum_{s=1}^{s_2^*}|S_s| \;\le\; \sum_{s=1}^{s_1^*} t + \sum_{s=s_1^*+1}^{s_2^*} t\left(\sqrt{\tfrac{2}{3}}\right)^{s-s_1^*-1} \;\le\; ts_1^* + t\sum_{s=0}^{\infty}\left(\sqrt{\tfrac{2}{3}}\right)^{s} \\ &\le\; ts_1' + t\,\frac{1}{1-\sqrt{\frac{2}{3}}} \;=\; t\left(\left\lceil \log_2\left(\frac{\sigma^2 n_0}{t^2}\right)\right\rceil + 3 + \sqrt{6}\right), \end{aligned}$$

where (3.5) was used to replace s_1'. Consequently, the constructed m-cut (B, W) in G_0 satisfies the desired bound on the width by Lemma 3.2.

To analyze the running time of the algorithm following the above construction, note that the first phase and the second phase can be implemented as the algorithms contained in Theorem 3.8 and in Theorem 3.5, respectively. There is only one implementation of the algorithm in Theorem 3.5, which takes $\mathcal{O}(n')$ time when given a planar graph on n' vertices as input. Here, if the second phase is entered, the algorithm receives the graph $G_{s_1^*}$ as input. Using that $G_{s_1^*}$ is a subgraph of G_0, it follows that the second phase takes $\mathcal{O}(n_0)$ time.

For the first phase, consider first the implementation of the algorithm contained in Theorem 3.8 that takes $\mathcal{O}\left((\|G'\| + \|(T', \mathcal{X}')\|)\log_2(n')\right)$ time when given a graph G' on n' vertices and a tree decomposition $(T', \mathcal{X}')$ of G' as input. In the proof of Theorem 3.8, it was argued that one execution of the while loop takes $\mathcal{O}\left(\|G'\| + \|(T', \mathcal{X}')\|\right)$ time. Here, the input graph G_0 is planar and therefore $|E(G_0)| \le 3n_0$ by Corollary 2.9, which implies that $\|G_0\| \le 4n_0 \le \mathcal{O}(\|(T_0, \mathcal{X}_0)\|)$. Furthermore, the first phase consists of at most s_1' executions of the while loop. So the first phase takes $\mathcal{O}(s_1' \cdot \|(T_0, \mathcal{X}_0)\|)$ time and therefore the entire running time is $\mathcal{O}(s_1' \cdot \|(T_0, \mathcal{X}_0)\|)$. Together with (3.5) the first bound on the running time follows. Observe that, as $s_1' = 0$ when $t = \sigma\sqrt{n_0}$, the running time uses the factor $\log\left(\frac{2\sigma^2 n_0}{t^2}\right) \ge s_1'$, which is always positive.

Now, consider the second implementation of the algorithm in Theorem 3.8, which takes $\mathcal{O}(n't')$ time when given a nonredundant tree decomposition of a graph on n' vertices of width t' as input. So if $(T_0, \mathcal{X}_0)$ is nonredundant, at most $\mathcal{O}(n_0 t)$ time is needed for the first phase. Using that the second phase needs at most $\mathcal{O}(n_0)$ time yields the second bound on the running time. □

Observe that for $t > \sigma\sqrt{n}$ the strategy used in the proof of Theorem 3.9 results in Theorem 3.5 as $s_1' \le 0$ and the first phase is not entered.

At the end of Section 3.1.3 it was mentioned that there is an algorithm similar to the one contained in Theorem 3.8 that runs in linear time. This algorithm can replace the first phase of the algorithm contained in the proof of Theorem 3.9 such that a cut with the properties in Theorem 3.9 can also be computed in linear time, see Corollary 4.11 in Chapter 4.

3.2 Planar Graphs with Large Minimum Bisection Width

This section focuses on bounded-degree planar graphs and discusses the relations of the following three properties: having large minimum bisection width, having large tree-width, and containing a large grid as a minor. Section 3.2.1 clarifies what is meant by each of these properties, and points out some relationships. For bounded-degree planar graphs, the only property that is not implied by one of the other two is large minimum bisection width. In Section 3.2.2, different ideas on how a bounded-degree planar graph, that contains a large grid as a minor, can be forced to have large minimum bisection width are discussed and the concept of grid-homogeneous graphs is introduced. Section 3.2.3 shows that, roughly speaking, grid-homogeneous graphs have large minimum bisection width.

3.2.1 Minimum Bisection Width, Tree-Width, and Grid Minors in Planar Graphs

Before discussing the relationship of these parameters, let us specify what large minimum bisection width, large tree-width, and containing a large grid as a minor refer to. To do so, the next proposition states upper bounds for these parameters and the proposition after shows that each of them is tight up to a constant factor. To quickly repeat some notation, consider a graph $G = (V, E)$. Recall that $\text{MinBis}(G)$ denotes the width of a minimum bisection in G, i. e.,

$$\text{MinBis}(G) := \min \{e_G(B, W)\colon \ \ (B, W) \text{ is a bisection in } G\}.$$

Furthermore, $\text{tw}(G)$ denotes the tree-width of G, which is the smallest integer $t \in \mathbb{N}_0$ such that G allows a tree decomposition of width t. Recall the definition of a $k \times k$ grid in Section 2.2, i. e., for $k \in \mathbb{N}$ the $k \times k$ grid is the graph $G_k = (V_k, E_k)$ with

$$V_k = \{v = (i, j) : \ i \in [k], j \in [k]\} \qquad \text{and}$$

$$E_k = \left\{ \{v, v'\} \in \binom{V_k}{2} \colon \ v = (i, j)\,, v' = (i', j')\,, \text{ and } |i - i'| + |j - j'| = 1 \right\}.$$

For grid minors, define

$$\text{grid}(G) := \max \{k \in \mathbb{N}\colon \ \ G \text{ contains a } k \times k \text{ grid as a minor}\}.$$

Proposition 3.10.
Fix an arbitrary $\Delta_0 \in \mathbb{N}$. Every planar graph G on n vertices with $\Delta(G) \leq \Delta_0$ satisfies
a) $\text{MinBis}(G) = \mathcal{O}(\sqrt{n})$,
b) $\text{tw}(G) = \mathcal{O}(\sqrt{n})$, *and*
c) $\text{grid}(G) = \mathcal{O}(\sqrt{n})$.

Proof. As in the statement, fix an arbitrary $\Delta_0 \in \mathbb{N}$. Consider an arbitrary planar graph G on n vertices with $\Delta(G) \leq \Delta_0$.
a) This part follows directly from Theorem 1.10.
b) This is the same as Proposition 2.13, which is stated in [Bod98].
c) As a $k \times k$ grid contains k^2 vertices, also every graph that contains a $k \times k$ grid as a minor needs to contain at least k^2 vertices. Hence, $\text{grid}(G) \leq \sqrt{n}$. □

Observe that in the previous proposition the assumption that the degree of the considered graph is bounded is only necessary for Part a). Indeed, the star $K_{1,n-1}$ on n vertices satisfies $\Delta(K_{1,n-1}) = n - 1$, is planar, and does not allow a bisection of width less than $\frac{1}{2}(n - 1)$. Furthermore, when the graph G is not required to be planar anymore, then the bounds in Part a) and Part b) do not hold, as for example the complete graph K_n on n vertices satisfies $\text{tw}(K_n) = n - 1$ and $\text{MinBis}(K_n) = \frac{1}{4}n^2$ when n is even.

Proposition 3.11.
Let G be a $k \times k$ grid with $k \geq 2$. Then

a) $\operatorname{MinBis}(G) \geq k$, $\operatorname{MinBis}(G) = k$ *if k is even,*
b) $\operatorname{tw}(G) = k$, *and*
c) $\operatorname{grid}(G) = k$.

Proof. Fix an integer $k \geq 2$ and let $G = (V, E)$ be the $k \times k$ grid.

a) The lower bound $\operatorname{MinBis}(G) \geq k$ was stated in Corollary 2.7. If k is even, equality holds since the bisection (B, W) with $B := \left\{(i, j) : i \in \left[\frac{1}{2}k\right], j \in [k]\right\}$ and $W = V \setminus B$ has width k.
b) See Proposition 2.12b).
c) Obvious. □

Consider a family $\mathcal{G}$ of bounded-degree planar graphs. We say that $\mathcal{G}$ is a family of bounded-degree planar graphs with *large minimum bisection width* if there is a constant $c > 0$ such that every graph $G \in \mathcal{G}$ satisfies $\operatorname{MinBis}(G) \geq c\sqrt{|V(G)|}$, i. e., the minimum bisection width is as large as possible up to a constant factor. Similarly, if there is a constant $c' > 0$ such that $\operatorname{tw}(G) \geq c'\sqrt{|V(G)|}$ for all $G \in \mathcal{G}$, we say $\mathcal{G}$ is a family of bounded-degree planar graphs with *large tree-width* and, if there is a constant $c'' > 0$ such that $\operatorname{grid}(G) \geq c''\sqrt{|V(G)|}$ for all $G \in \mathcal{G}$, we say that $\mathcal{G}$ is a family of bounded-degree planar graphs containing a *large grid as a minor*. For better readability, we abuse notation when introducing ideas and use the terms "large minimum bisection width", "large tree-width", and "large grid minor" for single graphs. Consider a planar bounded-degree graph G. Proposition 3.10 roughly says that the upper bounds on the minimum bisection width, the tree-width, and the largest grid minor in G are the same. So it is natural to ask whether one of these parameters being large already implies that another one is large. The easiest direction is from containing a large grid as a minor to having large tree-width. Assume that G contains a $k \times k$ grid H as a minor. Then, Proposition 2.15 and Proposition 3.11b) imply $\operatorname{tw}(G) \geq \operatorname{tw}(H) \geq k$. The reverse direction is also true. Indeed, Robertson, Seymour, and Thomas [RST94] showed that $\operatorname{grid}(G) \geq \frac{1}{6}(\operatorname{tw}(G) + 4)$ for every planar graph G, which has been improved to $\operatorname{grid}(G) \geq \frac{1}{5}(\operatorname{tw}(G) + 5)$ for every planar graph G by Grigoriev [Gri11]. The next lemma summarizes this.

Lemma 3.12.
For every $c > 0$, there is a $\gamma'' = \gamma''(c) > 0$ such that the following is true for every planar graph G on n vertices.

a) If $\operatorname{grid}(G) \geq c\sqrt{n}$, *then* $\operatorname{tw}(G) \geq c\sqrt{n}$.
b) If $\operatorname{tw}(G) \geq c\sqrt{n}$, *then* $\operatorname{grid}(G) \geq \gamma''\sqrt{n}$.

So having large tree-width and containing a large grid as a minor is equivalent for planar graphs. To answer the question on how large minimum bisection width in bounded-degree planar graphs relates to these, Theorem 3.9 from the previous section is used to derive the following theorem, which was introduced in Section 1.2.

Theorem 3.13 (Theorem 1.4 restated).
For every $\Delta_0 \in \mathbb{N}$ and every $c > 0$, there is a $\gamma = \gamma(c, \Delta_0) > 0$ such that, for every planar graph G on n vertices with maximum degree at most Δ_0, the following holds

$$\operatorname{MinBis}(G) \;\geq\; c\sqrt{n} \qquad \Rightarrow \qquad \operatorname{tw}(G) \;\geq\; \gamma\sqrt{n} - 1.$$

$$\begin{tikzcd}[column sep=small]
 & \begin{array}{c}\mathrm{MinBis}(G) = \Omega(\sqrt{n}) \\ \text{for all } G \in \mathcal{G}\end{array} \arrow[dl, "\text{Theorem 3.13}"'] \arrow[dr, "\text{Corollary 3.14}"] & \\
 & \begin{array}{c}\mathcal{G} \text{ family of} \\ \text{bounded-degree planar} \\ \text{graphs, } n := |V(G)|\end{array} & \\
\begin{array}{c}\mathrm{tw}(G) = \Omega(\sqrt{n}) \\ \text{for all } G \in \mathcal{G}\end{array} \arrow[uur, rightsquigarrow, bend right, "?"'] \arrow[rr, shift left, "\text{Lemma 3.12b)}"] & & \begin{array}{c}\mathrm{grid}(G) = \Omega(\sqrt{n}) \\ \text{for all } G \in \mathcal{G}\end{array} \arrow[ll, shift left, "\text{Lemma 3.12a)}"] \arrow[uul, rightsquigarrow, bend left, "?"]
\end{tikzcd}$$

Figure 3.3: The relationship of large minimum bisection width, large tree-width, and containing a large grid as a minor in bounded-degree planar graphs.

Proof. Fix $\Delta_0 \in \mathbb{N}$, $c > 0$, and $\sigma := \sqrt{8}$. As $\log_2(x) \leq \sqrt{x}$ for all $x > 0$, we have

$$\gamma\left(\left\lceil 2\log_2\left(\frac{\sigma}{\gamma}\right)\right\rceil\right) \;\leq\; \gamma\left(2\frac{\sqrt{\sigma}}{\sqrt{\gamma}} + 1\right) \quad \xrightarrow{\gamma\to 0} \quad 0.$$

Therefore, it is possible to choose a $\gamma > 0$ with

$$\gamma\left(\left\lceil 2\log_2\left(\frac{\sigma}{\gamma}\right)\right\rceil + 3 + \sqrt{6}\right)\Delta_0 \;<\; c.$$

To show the contraposition of the claim, let G be an arbitrary planar graph on n vertices with maximum degree at most Δ_0 and $\mathrm{tw}(G) < \gamma\sqrt{n} - 1$. Consider a tree decomposition $(T, \mathcal{X})$ of G of width at most $t - 1 = \gamma\sqrt{n} - 1$. Theorem 3.9 applied to G and $(T, \mathcal{X})$ with size-parameter $m := \left\lfloor \frac{1}{2}n \right\rfloor$ implies that

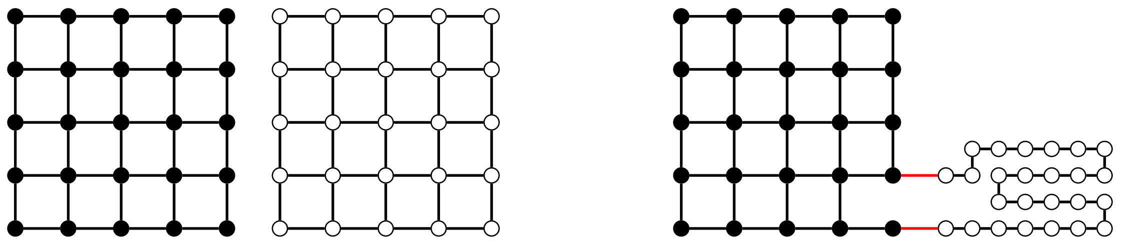
$$\mathrm{MinBis}(G) \;\leq\; \gamma\sqrt{n}\left(\left\lceil \log_2\left(\frac{\sigma^2}{\gamma^2}\right)\right\rceil + 3 + \sqrt{6}\right)\Delta_0 \;<\; c\sqrt{n}.$$ □

The previous theorem and Lemma 3.12b) immediately imply the following corollary.

Corollary 3.14 (Corollary 1.5 restated).
For every $\Delta_0 \in \mathbb{N}$ and every $c > 0$, there is a $\gamma' = \gamma'(c, \Delta_0) > 0$ such that, for every planar graph G on n vertices with maximum degree at most Δ_0, the following holds

$$\mathrm{MinBis}(G) \;\geq\; c\sqrt{n} \qquad \Rightarrow \qquad \mathrm{grid}(G) \;\geq\; \gamma'\sqrt{n}.$$

Figure 3.3 summarizes the results on large minimum bisection width, large tree-width, and containing a large grid as a minor in bounded-degree planar graphs. So the question whether large tree-width implies large minimum bisection width, or equivalently, whether containing a large grid as a minor implies large minimum bisection width, is not covered by the discussion so far. The answer to this is no, as Figure 3.4 shows two examples of bounded-degree planar graphs with large tree-width, but not large minimum bisection width, which can be easily extended to an infinitely large family. Figure 3.4a) shows a planar graph G_1 that consists of two copies of a $k_1 \times k_1$ grid. The number of vertices of G_1 is $n_1 = 2k_1^2$ and, hence, G_1 contains a large grid as a minor. Lemma 3.12a) implies that G_1 also has large tree-width. However, $\mathrm{MinBis}(G_1) = 0$ and G_1 does not have large minimum bisection width.



a) Graph G_1 for $k_1 = 5$ with $\mathrm{MinBis}(G_1) = 0$. **b)** Graph G_2 for $k_2 = 5$ with $\mathrm{MinBis}(G_2) = 2$.

Figure 3.4: Two bounded-degree planar graphs with large tree-width, that each allow a bisection of constant width. The vertices are colored black and white to show that each graph allows a bisection of small width. Edges cut by this bisection are colored red.

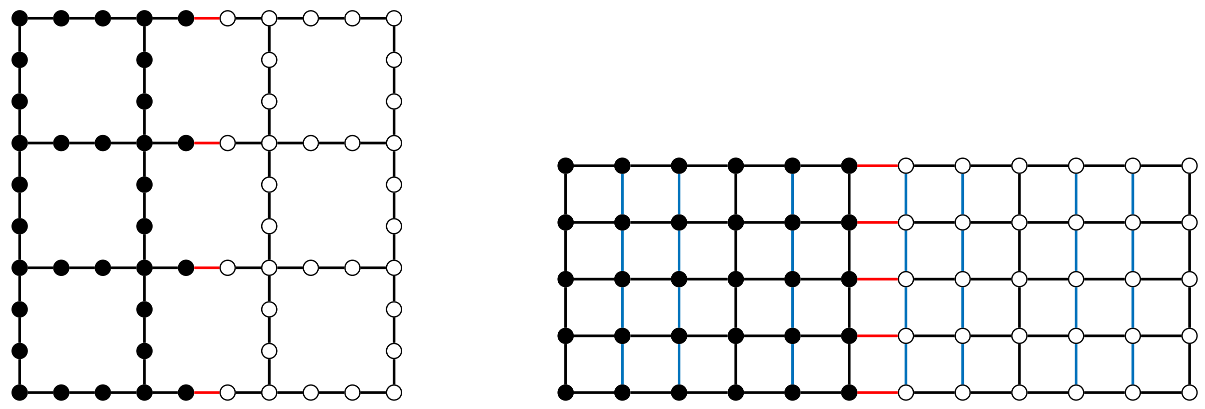
Figure 3.4b) shows a graph G_2, that is obtained from a $k_2 \times k_2$ grid by subdividing one edge k_2^2 times. Thus, G_2 has $n_2 = 2k_2^2$ vertices. Similar to G_1, the graph G_2 contains a large grid as a minor and has large tree-width, but $\mathrm{MinBis}(G_2) \leq 2$. The aim of Section 3.2.2 is to introduce a property such that all bounded-degree planar graphs that satisfy this property and contain a large grid as a minor also have large minimum bisection width.

3.2.2 Grid-Homogeneous Graphs

Consider a bounded-degree planar graph G that contains a large grid as a minor, or equivalently, that has large tree-width. The examples in Figure 3.4 show that G does not necessarily have large minimum bisection width. So what extra properties can we require to ensure that G must have large minimum bisection width? Here, two such properties are introduced. One of them is based on a large grid minor in G and the other one is based on many paths in G and can be generalized to go along with a tree decomposition of G of large width. Both properties are technical and long. So before stating them, some easier ideas are presented and it is discussed why they do not work. It is desirable that the $k \times k$ grid satisfies the property that together with containing a large grid as a minor implies large minimum bisection width.

Connectivity. One problem of the examples in Figure 3.4 seems to be the connectivity. For $s \in \mathbb{N}$, a graph G is *s-connected* if G contains at least s vertices and $G - S$ is connected for every set $S \subseteq V(G)$ with $|S| < s$. However, requiring the graph to be *s-connected* for some integer $s \in \mathbb{N}$ does not work here. First, any planar graph G contains a vertex v with $\deg_G(v) \leq 5$, as otherwise Corollary 2.9 was violated. Assuming that G contains at least 7 vertices, it follows that $G - S$ for $S = N_G(v)$ is not connected and G is not s-connected for every $s \geq 6$. As planar graphs are considered here, $s \geq 6$ does not make sense. Furthermore, the $k \times k$ grid G_k with $k \geq 2$ is not s-connected for every $s \geq 3$, because its corner vertex $v = (1, 1)$ has only two neighbors.

Many Paths. One of the properties of the $k \times k$ grid, that the graphs in Figure 3.4 do not have, is that there are many paths between medium sized vertex sets. More precisely the following is an idea for the property. Consider a bounded-degree planar graph $G = (V, E)$ on n vertices, and assume that there are two positive constants c and c' such that for all subsets $Z_1, Z_2 \subseteq V$ with $|Z_1| = |Z_2| = \lceil c\sqrt{n} \rceil$ and $Z_1 \cap Z_2 = \emptyset$ there are $c'\sqrt{n}$ edge-disjoint paths from Z_1 to Z_2. Let (B, W) be an arbitrary bisection in G and choose $Z_1 \subseteq B$ and $Z_2 \subseteq W$ with $|Z_1| = |Z_2| = \lceil c\sqrt{n} \rceil$. The property then implies that there are $c'\sqrt{n}$ edge-disjoint paths from Z_1 to Z_2. As each of these paths starts in B and ends in W, each path contains at least one edge that is cut by the bisection (B, W). Hence, $e_G(B, W) \geq c'\sqrt{n}$



a) Graph G_1 with $k_1 = 4$ and $\ell'_1 = 2$. **b)** Graph G_2 with $k_2 = 12$ and $k'_2 = 5$.

Figure 3.5: Bounded-degree planar graphs with large minimum bisection width. A subgraph, which can be contracted to a square grid and which is spread homogeneously through the graph, is obtained by deleting the blue edges. A minimum bisection (B, W) is indicated by the black and white vertices, edges cut by (B, W) are colored red.

and also $\text{MinBis}(G) \geq c'\sqrt{n}$. So G has large minimum bisection width. While this property implies the desired lower bound on the minimum bisection width, it is too strong as no assumption on the tree-width or a large grid minor is needed. Furthermore, when looking closely, the $k \times k$ grid does not satisfy this property. Indeed, let G_k be the $k \times k$ grid and $n_k := k^2$. Assume that $c^{\frac{1}{2}} n_k^{\frac{1}{4}}$ is an integer and consider the set

$$Z_1 := \left\{(i, j) \colon\ i \in \left[c^{\frac{1}{2}} n_k^{\frac{1}{4}}\right],\ j \in \left[c^{\frac{1}{2}} n_k^{\frac{1}{4}}\right]\right\},$$

which contains $c\sqrt{n_k}$ vertices. There are only $2c^{\frac{1}{2}} n_k^{\frac{1}{4}}$ edges in G_k that join a vertex in Z_1 to a vertex that is not in Z_1. Hence, at most $2c^{\frac{1}{2}} n_k^{\frac{1}{4}}$ edge-disjoint paths can start in Z_1 and G_k does not satisfy the property when k is large.

Large Grid as Subgraph. With Lemma 2.5 in mind, another idea for the property is to require that the graph contains a subgraph on more than half its vertices, which is a square grid. More precisely, fix $0 < \varepsilon \leq \frac{1}{2}$ and consider a graph G on n vertices that contains a subgraph H, which is isomorphic to the $k \times k$ grid for some $k \geq \sqrt{\left(\frac{1}{2} + \varepsilon\right) n}$. Fix a minimum bisection (B, W) in G, let $V' := V(H)$, and define $B' = B \cap V'$ as well as $W' = W \cap V'$. Without loss of generality assume that $|B'| \leq |W'|$ and for simplicity assume that n is even. As $|B| = |W| = \frac{1}{2}n$ and $n' := |V'| \geq \left(\frac{1}{2} + \varepsilon\right) n$, at least εn vertices of V' are in B, so $\varepsilon n' \leq \varepsilon n \leq |B'| \leq \frac{1}{2}n'$. Then, Lemma 2.5 applied to the grid H implies that

$$\text{MinBis}(G) \ \geq\ e_H(B', W') \ \geq\ \min\left\{\sqrt{\varepsilon},\, \tfrac{1}{2}\right\} \cdot \sqrt{\left(\tfrac{1}{2} + \varepsilon\right) n} \ =\ \Omega(\sqrt{n})$$

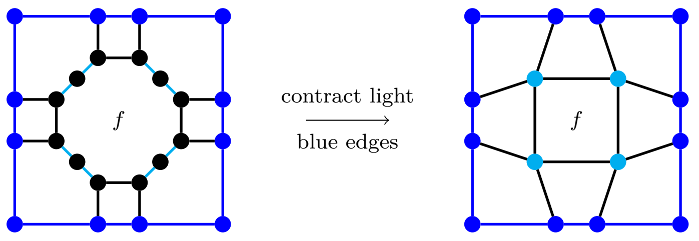
and, hence, G has large minimum bisection width. However, requiring a large square grid as a subgraph is a rather strong condition and it is not possible to loosen it by requiring a large subdivision of a square grid, as the example in Figure 3.4b) shows. Furthermore, Figure 3.5 displays two examples of bounded-degree planar graphs with large minimum bisection width that both do not contain a grid on more than half their vertices as subgraph. The graph G_1 in Figure 3.5a) is obtained from the $k_1 \times k_1$ grid by subdividing each edge $\ell'_1 \in \mathbb{N}$ times, where ℓ'_1 is a constant. Observe that G_1 does not contain a $k \times k$ grid as a subgraph for any $k \geq 2$. The graph G_2 in Figure 3.5b) shows a non-square grid, more precisely the $k_2 \times k'_2$ grid where $k_2 \geq 2k'_2$ and $k'_2 = \Theta(k_2)$. It is easy to see that the tree-width of G_2 is k'_2 by extending a tree decomposition of a $k_2' \times k_2'$ grid of width k_2', as the one in Section 2.3, to a tree decomposition of G_2. Let H be a $k_H \times k_H$ grid that is contained in G_2 as a subgraph, where k_H is arbitrary. Then, Proposition 3.11b) and Proposition 2.14 imply that $k_H \leq \operatorname{tw}(H) \leq \operatorname{tw}(G_2) \leq k_2'$. Observe that G_2 has $k_2 \cdot k_2'$ vertices and H has at most $(k_2')^2 \leq \frac{1}{2} k_2 \cdot k_2'$ vertices. So neither G_1 nor G_2 has a square grid on more than half its vertices as a subgraph. It is easy to see that $\operatorname{MinBis}(G_1) \leq k_1$ and $\operatorname{MinBis}(G_2) \leq k_2'$ when k_1 and k_2 are even. Showing that $\operatorname{MinBis}(G_1) = k_1$ and $\operatorname{MinBis}(G_2) = k_2'$ when k_1 and k_2 are even is harder and here, we will only show that $\operatorname{MinBis}(G_1) = \Omega(k_1)$ and $\operatorname{MinBis}(G_2) = \Omega(k_2')$ later. So, G_1 and G_2 have large minimum bisection width and, as each of them has a lot of structure, it is desirable that they satisfy the property that we want to find.

Consider the graphs in Figure 3.4 and Figure 3.5 once more. All graphs have a large grid as a minor and, hence, also have large tree-width. The ones in Figure 3.5, which have large minimum bisection width, have a large grid as a minor, which is, roughly speaking, spread homogeneously through the graph. This is not the case for the graphs in Figure 3.4, which do not have large minimum bisection width. In order to make this more precise, a few more definitions concerning planar graphs and grids are introduced. Recall the definition of the $k \times k$ grid from Section 2.2. The vertex set of the $k \times k$ grid is $\{(i,j) \colon i \in [k], j \in [k]\}$ and usually the vertex (i,j) is drawn at the point (i,j) of a coordinate system.

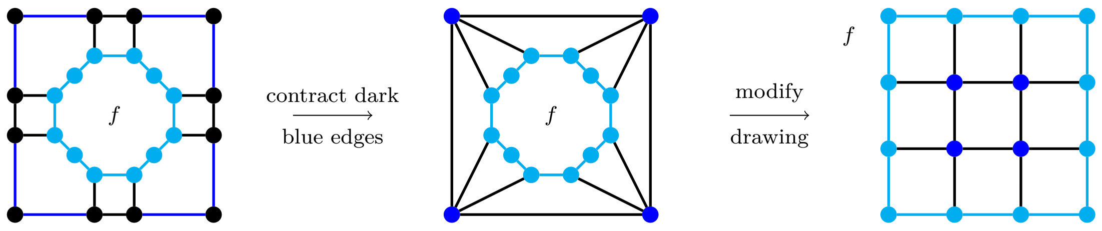
Recall that a graph is planar if it admits a drawing in the plane without crossing edges. A drawing with these properties is also called an *embedding in the plane* and a graph with a given embedding is called a *plane* graph. A planar graph G *is uniquely embeddable* if there is a topological isomorphism between any two embeddings of G in the plane, see Chapter 4.3 in [Die12] for further details. Roughly speaking, this means that the combinatorial structures of any two embeddings of G in the plane are the same, which can be made more precise with the cyclical orderings as follows. For any two embeddings $\mathcal{G}_1$ and $\mathcal{G}_2$ of G in the plane, either, for each vertex x of G, the cyclical ordering of x in $\mathcal{G}_1$ is identical to the cyclical ordering of x in G_2, or, for each vertex x of G, the cyclical ordering of x in $\mathcal{G}_2$ is obtained by reversing the cyclical ordering of x in $\mathcal{G}_1$. The latter one corresponds to reflecting the embedding. For $k \geq 2$ it is not hard to see that the $k \times k$ grid is uniquely embeddable. This is proved formally by using Whitney's Theorem and Tutte's Wheel Theorem in Appendix B. Let $k \geq 3$ and consider an embedding of the $k \times k$ grid G_k in the plane. All faces of G_k are bounded by cycles of length four, except for one face, which is bounded by a cycle of length $4(k-1) \geq 4 \cdot 2 = 8$. Let f be a face of G_k. If f is bounded by a cycle of length 4, then f is called a *small face* of G_k and otherwise f is called the *large face*. Next, this definition is extended to certain graphs containing a grid as a minor. A graph $H = (V_H, E_H)$ is a *minimal graph containing a $k \times k$ grid as a minor* if H contains G_k as a minor, $\operatorname{grid}(H - v) < k$ for all $v \in V_H$, and $\operatorname{grid}(H - e) < k$ for all $e \in E_H$. In other words, H is a minimal graph containing a $k \times k$ grid as a minor if deleting any vertex or any edge from H destroys the property of containing a $k \times k$ grid as a minor. The next remark extends the observation that a $k \times k$ grid with $k \geq 3$ is uniquely embeddable; a formal proof can be found in Appendix B.

Remark 3.15.
For every integer $k \geq 3$, every minimal graph containing a $k \times k$ grid as a minor is uniquely embeddable.

Consider a plane graph H which is a minimal graph containing a $k \times k$ grid as a minor for $k \geq 3$. Even though the embedding of H in the plane is essentially unique and there is a natural one-to-one correspondence between the faces of H and the faces of G_k when contracting H to obtain G_k and modifying the drawing accordingly, the definition of small and large faces cannot be extended directly to minimal graphs containing a $k \times k$ grid as a minor. For example, Figure 3.6 shows a graph H_1 that is a minimal graph containing a 4×4 grid as a minor and that can be contracted to the 4×4 grid G_4 in two ways, such



a) The face bounded by the dark blue cycle is the large face.



b) The face bounded by the light blue cycle is the large face.

Figure 3.6: A graph H that is a minimal graph containing a 4×4 grid as a minor and two ways to contract H to a 4×4 grid that result in different large faces of the 4×4 grid.

that the face f of H_1 can correspond to a small face of G_4 or to the large face of G_4. Let H be a minimal graph containing a $k\times k$ grid as a minor with $k \geq 5$. To uniquely define a large face, consider an embedding of H, fix one way to contract H to a grid G_k and consider the natural one-to-one correspondence between the faces of H and the faces of G_k. Each face f of H that corresponds to a small face of G_k can have at most eight vertices of degree 3 or greater on its boundary. Indeed, if there are two vertices u and v that are contracted to a vertex w with $\deg(w) < \deg(u) + \deg(v) - 2$, then there is an edge e that is incident to u and can be removed from H, such that $H - e$ contains a $k\times k$ grid as a minor, which is a contradiction. So each vertex of G_k with degree 4 results from contracting a tree $T \subseteq H$ into one vertex such that there are exactly four edges in H that join a vertex in T to a vertex not in T. Hence, T either contains two vertices u, v with $\deg_H(u) = \deg_H(v) = 3$ or one vertex v with $\deg_H(v) = 4$, and in both cases no other vertices of degree 3 or greater, i. e., T contains at most two vertices v with $\deg_H(v) \geq 3$. Similarly, each vertex of G_k with degree 3 results from contracting a tree $T \subseteq H$ into one vertex such that T contains exactly one vertex v with $\deg_H(v) \geq 3$ and each vertex of G_k with degree 2 results from contracting a tree $T \subseteq H$ into one vertex such that T contains no vertex v with $\deg_H(v) \geq 3$. Therefore, each face f of H that corresponds to a small face of G_k can have at most eight vertices of degree 3 or more on its boundary. It follows similarly that the face that corresponds to the large face of G_k must have exactly $4(k-2) \geq 12$ vertices of degree 3 on its boundary. Therefore, there is only one choice for the face of H that corresponds to the large face of G_k and the definitions of large and small faces can be extended to minimal graphs containing a large enough grid as a minor as follows. Let H be a plane graph that is a minimal graph containing a $k\times k$ grid as a minor with $k \geq 5$. The unique face of H with $4(k-2)$ vertices of degree 3 on its boundary is called the *large face* of H and each of the other faces of H is called a *small face* of H. The next propositions summarize the properties of small and large faces.

Proposition 3.16.
Let $k \geq 3$ and let $H = (V_H, E_H)$ be a minimal graph containing a $k\times k$ grid as a minor. Then V_H admits a partition into non-empty sets $M_{i,j}$ with $i, j \in [k]$ such that contracting $M_{i,j}$ to one vertex (i, j) results in the $k\times k$ grid G_k. Moreover, each such partition of V_H has the following properties.

- *For each vertex (i,j) of G_k with* $\deg_{G_k}((i,j)) = 4$*, either there are two vertices $v_1, v_2 \in M_{i,j}$ with* $\deg_H(v_1) = \deg_H(v_2) = 3$ *and all other vertices $w \in M_{i,j}$ satisfy* $\deg_H(w) = 2$*, or there is a vertex $v \in M_{i,j}$ with* $\deg_H(v) = 4$ *and all other vertices $w \in M_{i,j}$ satisfy* $\deg_H(w) = 2$*.*
- *For each vertex (i,j) of G_k with* $\deg_{G_k}((i,j)) = 3$*, there is a vertex $v \in M_{i,j}$ with* $\deg_H(v) = 3$ *and all other vertices $w \in M_{i,j}$ satisfy* $\deg_H(w) = 2$*.*
- *For each vertex (i,j) of G_k with* $\deg_{G_k}((i,j)) = 2$*, all vertices $w \in M_{i,j}$ satisfy* $\deg_H(w) = 2$*.*

In particular, each vertex v of H satisfies $2 \leq \deg_H(v) \leq 4$*.*

Proposition 3.17.
Let $k \geq 5$ and let H be a plane graph that is a minimal graph containing a $k \times k$ grid as a minor. Then, the following holds.

a) For every small face f of H, there are at most 8 vertices v in H that are on the boundary of f and satisfy $\deg_H(v) \geq 3$*.*

b) Each vertex v on the boundary of the large face of H satisfies $\deg_H(v) \in \{2,3\}$*. Furthermore, there are exactly $4(k-2)$ vertices v with* $\deg_H(v) = 3$ *on the boundary of the large face of H.*

Let $k \geq 5$ and let H be a minimal graph that contains a $k \times k$ grid as a minor. Furthermore, consider a planar graph G' that contains H as a subgraph. The *induced embedding* of H with respect to an embedding of G' is the embedding of H obtained from the embedding of G' by deleting all edges in $E(G') \setminus E(H)$ and all vertices in $V(G') \setminus V(H)$. In the following, we always assume that G' is embedded in such a way that, in the induced embedding of H, the large face of H is the unique infinite face of H. Let C be a cycle in G'. In the embedding of G', the cycle C divides the plane into two regions. The region containing the infinite face of G' is called the *outside* of C and the other region is called the *inside* of C. Now, we are ready to present the definition of grid-homogeneous graphs. The idea is to call a graph G grid-homogeneous if it contains subgraphs G' and H as the ones above, where G' is obtained from G by deleting few vertices and H is spread homogeneously through G'.

Definition 3.18 ((γ, k, ℓ)-grid-homogeneous, Definition 1.6 repeated).
Let $k, \ell \in \mathbb{N}$ with $k \geq 5$ and $0 \leq \gamma < 1$. A graph $G = (V,E)$ is called (γ, k, ℓ)-*grid-homogeneous* if it contains a connected planar graph $G' = (V', E') \subseteq G$ with $|V'| \geq (1-\gamma)|V|$ and a graph $H = (V_H, E_H) \subseteq G'$ as subgraph such that G' has an embedding in the plane with the following properties:

(H1) The graph H is a minimal graph containing a $k \times k$ grid as a minor.

(H2) For every small face f of the induced embedding of H, at most ℓ vertices from V' are embedded in the face f including the vertices on its boundary.

(H3) No vertex of $V' \setminus V_H$ is embedded in the large face f of the induced embedding of H.

To give some examples of grid-homogeneous graphs, consider the $k \times k$ grid G. To satisfy the definition, one can choose $G' = H = G$ and it follows that the $k \times k$ grid is $(0, k, 4)$-grid-homogeneous. Moreover, the graphs in Figure 3.5 are grid-homogeneous for certain parameters γ, k, and ℓ. Indeed, recall the graph G_1 in Figure 3.5a), which is obtained from a $k_1 \times k_1$ grid by subdividing each edge ℓ'_1 times. The graph G_1 is connected and planar, and it is a minimal graph containing a $k_1 \times k_1$ grid as a minor. So choose $G'_1 = H'_1 = G_1$. Then (H1) and (H3) are satisfied. For (H2) consider a small face f of the induced embedding of H. The face f is bounded by a cycle containing $4 + 4\ell'_1$ vertices and, hence, (H2) is satisfied for $\ell = 4 + 4\ell'_1$. So G_1 is $(0, k_1, 4 + 4\ell'_1)$-grid-homogeneous. The graph G_2 in Figure 3.5b) is a $k_2 \times k'_2$ grid, where $k_2 \geq 2k'_2$. Choose $G' = G$ and define $\ell'_2 = \left\lceil \frac{k_2 - k'_2}{k'_2 - 1} \right\rceil$. It is easy to see that G' contains a subdivision H of a $k'_2 \times k'_2$ grid with $V(H) = V(G')$ and where each horizontal edge of H

is subdivided at most ℓ'_2 times and vertical edges of H are not subdivided. An example for such a subgraph H is obtained by deleting the blue edges in Figure 3.5b). Observe that H is a minimal graph containing a $k'_2 \times k'_2$ grid as a minor and (H3) is satisfied. Consider a small face f of H. Then, there are at most $4 + 2\ell'_2$ vertices of G' embedded in f including the vertices on the boundary of f. Therefore, G_2 is $(0, k'_2, 4 + 2\ell'_2)$-grid-homogeneous.

All three examples of grid-homogeneous graphs are (γ, k, ℓ)-grid-homogeneous with $\gamma = 0$. So, why is the parameter γ included? Consider a (γ, k, ℓ)-grid-homogeneous graph G and let G' and H be subgraphs as in Definition 3.18. The parameter γ denotes the fraction of vertices that are not important for satisfying the properties (H1)-(H3), which only refer to G' and H. This is useful for the following reason. If G_1 is a bounded-degree planar graph with large minimum bisection width, then the graph G_2 obtained from G_1 by adding a constant number of isolated vertices will also have large minimum bisection width but cannot satisfy the definition with $\gamma = 0$ as this would require G_2 to be connected. Consider again the (γ, k, ℓ)-grid-homogeneous graph G and its subgraphs $G' = (V', E')$ and H. There is nothing known about the vertices in $V \setminus V'$. So in order to deduce some lower bound on the width of a minimum bisection in G, it is necessary to require that $\gamma < \frac{1}{2}$. Indeed, otherwise half of the vertices of the graph G might be isolated and not included in G', which then allows a bisection of width zero in G.

The aim of this section is to show the following lower bound on the minimum bisection width in grid-homogeneous planar graphs.

Theorem 3.19 (Theorem 1.7 repeated).
For every $k, \ell \in \mathbb{N}$ with $k \geq 5$ and every $0 \leq \gamma < \frac{1}{2}$, every graph $G = (V, E)$ with $|V|$ even that is (γ, k, ℓ)-grid-homogeneous satisfies

$$\mathrm{MinBis}(G) \;\geq\; \left(\tfrac{1}{2} - \gamma\right) \frac{1}{4\ell} k.$$

Consider the three examples of grid-homogeneous graphs once more. Theorem 3.19 implies that the $k \times k$ grid G satisfies $\mathrm{MinBis}(G) \geq \frac{1}{2} \cdot \frac{1}{16} k = \frac{1}{32} k$ as it is $(0, k, 4)$-grid-homogeneous. Furthermore, it was argued that the graphs G_1 and G_2 in Figure 3.5 are $(0, k_1, 4 + 4\ell'_1)$-grid-homogeneous and $(0, k'_2, 4 + 2\ell'_2)$-grid-homogeneous, respectively. Therefore,

$$\mathrm{MinBis}(G_1) \;\geq\; \frac{1}{32(\ell'_1 + 1)} k_1 \qquad \text{and} \qquad \mathrm{MinBis}(G) \;\geq\; \frac{1}{16(\ell'_2 + 2)} k'_2,$$

which shows that the graphs G_1 and G_2 have large minimum bisection width as claimed above.

Before presenting the strategy to prove the theorem, let us have a closer look at the definition of grid-homogeneous graphs and see why none of the properties required there can be dropped in order to prove a lower bound on the minimum bisection width as in the previous theorem.

Connectivity of $\mathbf{G'}$. Let $k \geq 5$, let H be the $k \times k$ grid, and let G be the graph obtained by adding k^2 isolated vertices to H. Finally, let $G' = G$. Consider an embedding of H in the plane. As H has $(k-1)^2$ small faces and G' contains $k^2 < 2(k-1)^2$ isolated vertices, the embedding of H can be extended to an embedding of G' by drawing up to two isolated vertices in each small face of H, see Figure 3.7a). Then, Definition 3.18 is satisfied for $\gamma = 0$ and $\ell = 6$, except for the connectivity of G'. Choosing $B = V(H)$ and $W = V(G) \setminus B$ defines a bisection (B, W) in G and shows that $\mathrm{MinBis}(G) = 0$.

Property (H2). Fix $k \geq 5$ and let G be the graph obtained from a $k \times k$ grid by subdividing one of its edges k^2 times, see Figure 3.4b). Then, G is a minimal graph containing a $k \times k$ grid as a minor. If (H2) is neglected, then Definition 3.18 is satisfied with $\gamma = 0$ and $G' = H = G$. As argued in the end of Section 3.2.1, G allows a bisection of width 2. Furthermore, this example shows that the vertices on the boundary of each small face are important for (H2).

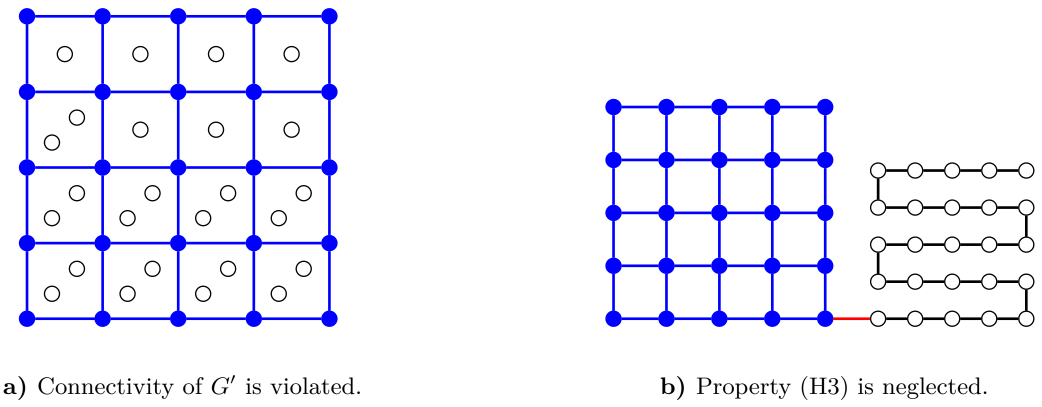

a) Connectivity of G' is violated.
b) Property (H3) is neglected.

Figure 3.7: Bounded-degree planar graphs that almost satisfy Definition 3.18. In both cases, $G = G'$ and the subgraph H is colored blue. The graphs admit bisections of width zero and one, respectively. A corresponding bisection is indicated by the solid and non-solid vertices, the cut edge in Part b) is colored red.

Property (H3). Fix $k \geq 3$ and consider a $k \times k$ grid H that is embedded in the plane. Let $G = G'$ be the graph obtained by joining a path P on k^2 vertices to one of the vertices on the boundary of the large face of H. Consider the embedding of $G' = G$ where P is embedded in the large face of H, see Figure 3.7b). Then, Definition 3.18 is satisfied with $\gamma = 0$ and $\ell = 4$ except for (H3). Choosing $B = V(H)$ and $W = V(G) \setminus B$ defines a bisection (B, W) in G and shows that $\mathrm{MinBis}(G) = 1$.

Observe that not every bounded-degree planar graph G with large minimum bisection width satisfies Definition 3.18 for some $\gamma > \frac{1}{2}$. For example, consider the planar graph G that is composed of three copies of the $k \times k$ grid for some even integer $k \in \mathbb{N}$ and contains $n = 3k^2$ vertices. Let (B, W) be a minimum bisection in G and let b_i for $i \in [3]$ be the number of vertices of B that are in each component of G. Without loss of generality assume that $b_1 \leq b_2 \leq b_3$. Clearly, $b_1 + b_2 + b_3 = \frac{3}{2}k^2$. Furthermore, $b_3 \leq k^2$ implies that $b_1 + b_2 \geq \frac{1}{2}k^2$ and hence $b_2 \geq \frac{1}{4}k^2$. Also, $b_2 + b_3 \leq \frac{3}{2}k^2$ and hence $b_2 \leq \frac{3}{4}k^2$. All in all, $\frac{1}{4}k^2 \leq b_2 \leq \frac{3}{4}k^2$. Let (B_2, W_2) be the cut in the second component of G that the bisection (B, W) defines. To estimate the edges cut by (B_2, W_2), one can assume without loss of generality that $|B_2| \leq |W_2|$ as otherwise the sets B_2 and W_2 can be switched in the following argument. So, $\frac{1}{4}k^2 \leq |B_2| \leq \frac{1}{2}k^2$ and Lemma 2.5 implies that at least $\frac{1}{2}k$ edges are cut in the second component. Therefore, $e_G(B, W) \geq \frac{1}{2}k = \frac{1}{2\sqrt{3}}\sqrt{n}$ and G has large minimum bisection width. However, as G does not have a connected subgraph on more than half its vertices, it cannot satisfy Definition 3.18 for any $\gamma > \frac{1}{2}$.

The proof of Theorem 3.19 is done in two steps. First, a large subgraph of the grid-homogeneous graph is partitioned, such that there are many paths between two sets of the partition. This is made more precise by the concept of path-prosperous graphs, which is introduced next. The second step of proving Theorem 3.19 is then to show a lower bound on the width of a minimum bisection in path-prosperous graphs. To define path-prosperous graphs, the following notation is used. For a graph $G = (V, E)$ and two (not necessarily disjoint) sets $A, B \subseteq V$, a path $P = (v_0, v_1, \ldots, v_\ell)$ in G is an *A,B-path*, if $v_0 \in A$, $v_\ell \in B$, and, if $\ell \geq 2$, then $v_h \notin A \cup B$ for all $h \in [\ell - 1]$. Note that every vertex in $A \cap B$ is an A,B-path in G.

Definition 3.20 ((γ, k, c)-path-prosperous).
Let $0 \leq \gamma < 1$, $k \in \mathbb{N}$, and $0 < c \leq 1$. A graph $G = (V, E)$ is called (γ, k, c)-path-prosperous if it contains a subgraph $G' = (V', E')$ with $|V'| \geq (1 - \gamma)|V|$ and such that there is a collection of clusters $\mathcal{X} = (X^i)_{i \in I}$ with $X^i \subseteq V'$ for each $i \in I$ satisfying the following properties

(P1) every vertex in V' is in precisely one set X^i with $i \in I$,

(P2) for all $i \in I$ the set X^i satisfies $|X^i| \geq k$, (P3) for all $d \in [k]$, for all $i_1, i_2 \in I$, and for all $Z_1 \subseteq X^{i_1}$, $Z_2 \subseteq X^{i_2}$ with $|Z_1| = |Z_2| = d$, there exist at least cd edge-disjoint Z_1,Z_2-paths in G'.

Consider a (γ, k, c)-path-prosperous graph $G = (V, E)$ with small γ. As before, the parameter γ denotes the fraction of vertices of G that are not important for satisfying (P1)-(P3). The graph G' in Definition 3.20 needs to be connected to satisfy (P3). Furthermore, when ignoring (P3), it is easy to simultaneously satisfy (P1) and (P2), e.g. by using one cluster that contains all vertices in V'. However, a large bounded-degree planar graph G' will not satisfy (P3) when $k = n' = |V(G')|$ and when choosing $\mathcal{X}$ to consist of only the cluster $V' := V(G')$. Indeed, according to Theorem 1.10 the graph G' allows a bisection (B', W') of width $\mathcal{O}(\sqrt{n'})$, so there are only $\mathcal{O}(\sqrt{n'})$ paths between the sets B' and W', which have size $\Theta(n')$. Roughly speaking, the larger the size of the clusters due to (P2), the harder it becomes to satisfy (P3).

Next, two lemmas are presented, which together immediately imply Theorem 3.19.

Lemma 3.21.
For every $k, \ell \in \mathbb{N}$ with $k \geq 5$ and every $0 \leq \gamma < 1$, the following holds. If G is a (γ, k, ℓ)-grid-homogeneous graph, then G is $(\gamma, k, \frac{1}{4\ell})$-path-prosperous.

Lemma 3.22.
For every $0 \leq \gamma < \frac{1}{2}$, for every $k \in \mathbb{N}$, and for every $0 < c \leq 1$, every (γ, k, c)-path-prosperous graph $G = (V, E)$ with $|V|$ even satisfies

$$\operatorname{MinBis}(G) \;\geq\; \left(\tfrac{1}{2} - \gamma\right) \cdot ck.$$

The concept of path-prosperous graphs presented here is rather restricted. Especially when thinking about the collection of clusters of a tree decomposition $(T, \mathcal{X})$ with $\mathcal{X} = (X^i)_{i \in V(T)}$, then (P1) seems too restrictive. Indeed, assume that $\mathcal{X}$ satisfies (P1). Then, due to Property (T2) of tree decompositions, the underlying graph G' decomposes into the disjoint parts $G'[X^i]$ for $i \in V(T)$, which means that G' and $\mathcal{X}$ cannot satisfy (P3). In Appendix A, a generalization of Definition 3.20 that relaxes the condition in (P1) is presented and a lower bound for the minimum bisection width of such graphs is derived.

3.2.3 Proof of the Lower Bound for Grid-Homogeneous Graphs

One technical detail in the proof of Lemma 3.21 is to find many disjoint paths between two columns of a grid. To do so, the following lemma is used. Note that there are many results on disjoint paths in grids, regarding vertex-disjoint paths as well as edge-disjoint paths. However, for the proof of Lemma 3.21 some special additional properties are needed.

Lemma 3.23.
Consider the $k \times k$ grid G_k for an arbitrary $k \in \mathbb{N}$ with $k \geq 8$. For $i \in [k]$, denote by C_i the i^{th} column of G_k and, for $j \in [k]$, denote by R_j the j^{th} row of G_k. For every $i_1, i_2, d \in [k]$ with $i_1 \leq i_2$, for every $Z_1 \subseteq C_{i_1}$, and for every $Z_2 \subseteq C_{i_2}$ with $|Z_1| \geq d$ and $|Z_2| \geq d$, there are $\lceil \frac{d}{2} \rceil$ vertex-disjoint Z_1,Z_2-paths in G_k

- *that do not use any vertical edges in C_i for all $i \in \{i_1, i_1 + 1, i_2, i_2 + 1\} \cap [k]$ and*
- *that do not use any horizontal edges in R_j for all $j \in [k]$ with $R_j \cap (Z_1 \cup Z_2) = \emptyset$.*

Proof. Fix $k \in \mathbb{N}$ with $k \geq 8$ and $i_1, i_2, d \in [k]$ with $i_1 \leq i_2$. Let $G = (V, E)$ be the $k \times k$ grid and choose two arbitrary sets $Z_1 \subseteq C_{i_1}$ and $Z_2 \subseteq C_{i_2}$ satisfying $|Z_1| \geq d$ and $|Z_2| \geq d$. Let $d' = \lceil \frac{d}{2} \rceil$, which is the number of Z_1,Z_2-paths that we want to find. Set

$$I_f = \{i_1, i_1 + 1, i_2, i_2 + 1\} \cap [k] \qquad \text{and} \qquad J_f = \{j \in [k] \colon\ R_j \cap (Z_1 \cup Z_2) = \emptyset\},$$

which are the indices of the columns and rows in which we need to avoid the vertical and horizontal edges, respectively, when constructing the paths. Define

$$d_1 := \begin{cases} 0 & \text{if } i_1 \leq 1 \\ \min\{i_1 - 1,\ d'\} & \text{otherwise,} \end{cases}$$

$$d_2 := \begin{cases} 0 & \text{if } i_2 + 1 \geq k \\ \min\{k - (i_2 + 1),\ d' - d_1\} & \text{otherwise,} \end{cases}$$

$$d_3 := \begin{cases} 0 & \text{if } i_2 \leq i_1 + 2 \\ \min\{(i_2 - 1) - (i_1 + 1),\ d' - (d_1 + d_2)\} & \text{otherwise.} \end{cases}$$

The strategy of the proof is to construct

- d_1 paths that use only vertical edges in columns C_i with $i \in [i_1 - 1]$,
- d_2 paths that use only vertical edges in columns C_i with $i \in [k] \setminus [i_2 + 1]$, and
- d_3 paths that use only vertical edges in columns C_i with $i \in [i_2 - 1] \setminus [i_1 + 1]$,

such that all paths are Z_1,Z_2-paths and they are pairwise vertex-disjoint. Furthermore, each path that joins a vertex $v \in Z_1$ to a vertex $w \in Z_2$ will only use horizontal edges in the row that contains v and the row that contains w. As

$$(i_1 - 1) + (k - (i_2 + 1)) + ((i_2 - 1) - (i_1 + 1)) \;=\; k - 4 \;\geq\; \left\lceil \tfrac{k}{2} \right\rceil \;\geq\; d',$$

the construction yields exactly $d' = d_1 + d_2 + d_3$ vertex-disjoint Z_1,Z_2-paths in the end.

Choose $j^* \in [k]$ such that $|Z_1^{\text{top}}| \geq d'$ as well as $|Z_1^{\text{bottom}}| \geq d'$ holds, where

$$Z_1^{\text{top}} := \{v = (i_1, j) \in Z_1 \colon\ j \geq j^*\} \qquad \text{and} \qquad Z_1^{\text{bottom}} := \{v = (i_1, j) \in Z_1 \colon\ j \leq j^*\}\,.$$

Such a j^* must exist as $Z_1^{\text{top}} \cup Z_1^{\text{bottom}} = Z_1$ and $Z_1^{\text{top}} \cap Z_1^{\text{bottom}} \subseteq \{(i_1, j^*)\}$ is nonempty for certain values of j^*. Furthermore, define

$$Z_2^{\text{top}} := \{w = (i_2, j) \in Z_2 \colon\ j \geq j^*\} \qquad \text{and} \qquad Z_2^{\text{bottom}} := \{w = (i_2, j) \in Z_2 \colon\ j \leq j^*\}\,.$$

As $Z_2^{\text{top}} \cup Z_2^{\text{bottom}} = Z_2$, we have $|Z_2^{\text{top}}| \geq d'$ or $|Z_2^{\text{bottom}}| \geq d'$. Without loss of generality, assume that $|Z_2^{\text{top}}| \geq d'$, because otherwise the grid can be flipped along a horizontal axis and the vertices can be relabeled accordingly. Choose $Z_1' \subseteq Z_1^{\text{bottom}}$ with $|Z_1'| = d'$ and $Z_2' \subseteq Z_2^{\text{top}}$ with $|Z_2'| = d'$. Label the vertices in Z_1' with $v_1 = (i_1, j_1^v)$, $v_2 = (i_1, j_2^v)$, …, $v_{d'} = (i_1, j_{d'}^v)$ such that $j_1^v < j_2^v < \ldots < j_{d'}^v$ and label the vertices in Z_2' with $w_1 = (i_2, j_1^w)$, $w_2 = (i_2, j_2^w)$, …, $w_{d'} = (i_2, j_{d'}^w)$ such that $j_1^w > j_2^w > \ldots > j_{d'}^w$.

Recall that $[0] = \emptyset$. For every $h \in [d_1]$, connect v_h and w_h with a path P_h that uses only edges in $R_{j_h^v}$, C_h, and $R_{j_h^w}$. It is easy to see that P_h and $P_{\hat{h}}$ do not intersect for distinct $h, \hat{h} \in [d_1]$, see the example in Figure 3.8. As $d_1 = 0$ or $d_1 \leq i_1 - 1$, no edges in C_i are used for all $i \in I_f$. Every path P_h with $h \in [d_1]$ uses exactly one vertex in Z_2, but it can happen that there is an $h \in [d_1]$ such that the path P_h uses two vertices from Z_1. In this case, the path P_h is shortened such that it uses exactly one vertex from Z_1, an example for this is path P_1 in Figure 3.8.

For every $h \in [d_2 + d_1] \setminus [d_1]$, set $h' = h - d_1$, and connect v_h and w_h with a path P_h that uses only edges in $R_{j_h^v}$, $C_{k-(h'-1)}$, and $R_{j_h^w}$. It is easy to see that P_h and $P_{\hat{h}}$ do not intersect for distinct $h, \hat{h} \in [d_1 + d_2]$. As $k - (d_2 - 1) \geq i_2 + 2$ or $d_2 = 0$, no vertical edges in C_i are used for all $i \in I_f$. Every path P_h with $h \in [d_1 + d_2] \setminus [d_1]$ uses exactly one vertex in Z_1, but it can happen that there is an $h \in [d_2 + d_1] \setminus [d_1]$ such that the path P_h uses two vertices from Z_2. In this case, the path P_h is shortened such that it uses exactly one vertex from Z_2, an example for this is the path P_4 in Figure 3.8.

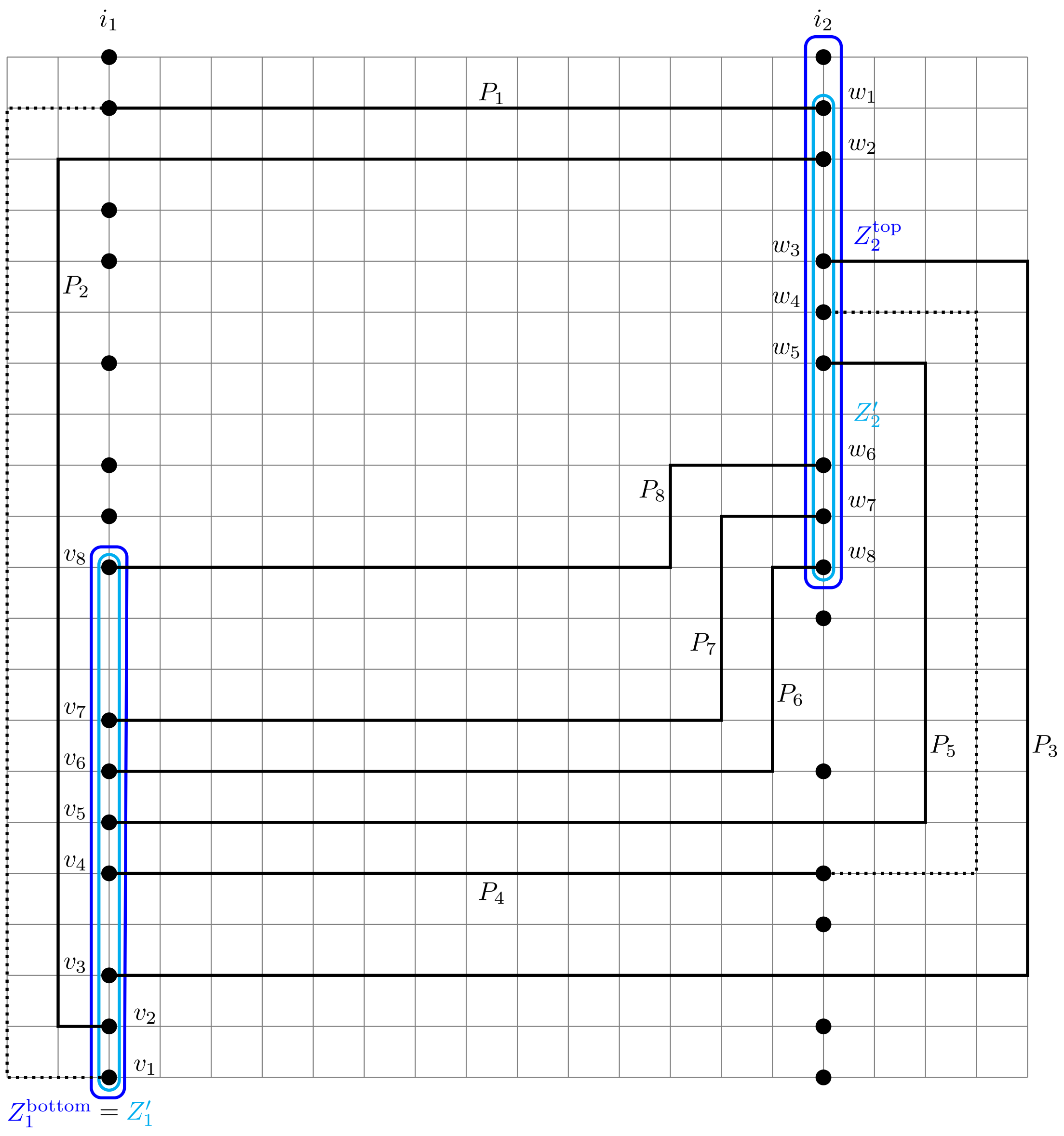


Figure 3.8: Paths used in the proof of Lemma 3.23. Example with $k = 21$, $d = 15$, $d' = 8$ and $j^* = 11$. Here, $d_1 = 2$, $d_2 = 3$, and $d_3 = 3$. Only the vertices in the sets Z_1 and Z_2 are drawn explicitly. Note that the paths P_1 and P_4 had to be shortened as they used two vertices in Z_1 and two vertices in Z_2, respectively.

For every $h \in [d'] \setminus [d_1 + d_2]$ set $h' = d' - (h - d_1 - d_2) + 1$. Connect v_h and $w_{h'}$ with a path P_h that uses only edges in $R_{j_h^v}$, $C_{i_2-(h-d_1-d_2)}$, and $R_{j_{h'}^w}$. It is easy to see that P_h and $P_{\hat{h}}$ do not intersect for distinct $h, \hat{h} \in [d']$. As for every $h \in [d'] \setminus [d_1 + d_2]$ we have $i_2 - (h - d_1 - d_2) \geq i_2 - d_3 \geq i_1 + 2$ or $d_3 = 0$, no vertical edges in C_i are used for all $i \in I_f$. Every path P_h with $h \in [d'] \setminus [d_1 + d_2]$ uses exactly one vertex in Z_1 and exactly one vertex in Z_2. □

Observe that the proof of Lemma 3.21 requires to show that, roughly speaking, there are many edge-disjoint paths in a graph G' that contains the $k \times k$ grid as a minor. Lemma 3.23 can be applied to find vertex-disjoint paths in the $k \times k$ grid and a version with edge-disjoint paths in the $k \times k$ grid would not suffice as the example in Figure 3.9 shows.

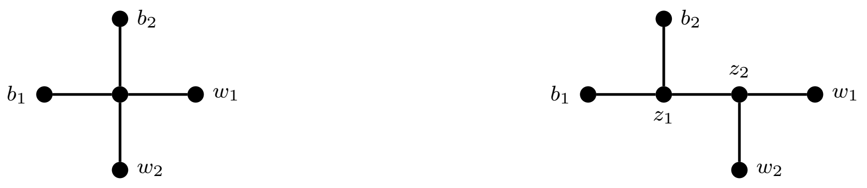


Figure 3.9: An example showing that a version of Lemma 3.23 with edge-disjoint paths does not suffice for the proof of Lemma 3.21. When contracting the edge $\{z_1, z_2\}$ in the right graph, the graph on the left is obtained. The graph on the left contains a b_1,w_1-path and a b_2,w_2-path that are edge-disjoint, but the graph on the right does not.

Proof of Lemma 3.21. Fix $0 \le \gamma < 1$ and $k, \ell \in \mathbb{N}$ with $k \ge 5$. Let $G = (V, E)$ be an arbitrary (γ, k, ℓ)-grid-homogeneous graph. Then, there are two subgraphs $G' \subseteq G$ and $H \subseteq G'$, as well as a plane embedding of G' with the properties required by Definition 3.18. Let $G' = (V', E')$ and $H = (V_H, E_H)$. Then $|V'| \ge (1-\gamma)|V|$ and (H1)-(H3) are satisfied. The aim is to show that G is $(\hat{\gamma}, \hat{k}, \hat{c})$-path-prosperous with $\hat{\gamma} = \gamma$, $\hat{k} = k$, and $\hat{c} = \frac{1}{4\ell}$. To do so, a partition of V' is constructed that satisfies the properties required by Definition 3.20. Roughly speaking, the partition is obtained by slicing the graph G into k pieces along the columns of H.

From now on, whenever an embedding of H or a face of the graph H is considered, we refer to the induced embedding of H with respect to the embedding of G' that satisfies the properties required by Definition 3.18. Without loss of generality, assume that G' satisfies the following property.

(H4) The graph $G' - e$ is disconnected for every $e \in E' \setminus E_H$. In particular, no edge in $E' \setminus E_H$ joins two vertices in V_H.

Indeed, if G' does not satisfy (H4), then one can successively delete such edges from G' without violating (H1)-(H3) or the connectivity of G'.

Let $\tilde{H} = (\tilde{V}_H, \tilde{E}_H)$ be the $k \times k$ grid as defined in Section 2.2. Denote by $\tilde{C}_i$ its i^{th} column for every $i \in [k]$ and by $\tilde{R}_j$ its j^{th} row for every $j \in [k]$. During this proof, variables with tilde are used to denote subgraphs and similar structures of the $k \times k$ grid $\tilde{H}$. Since H contains a $k \times k$ grid as a minor, there is a function $M : \tilde{V}_H \to \{X\colon\ X \subseteq V_H, X \neq \emptyset\}$ such that $M(i,j) \cap M(i',j') = \emptyset$ for all $i, j, i', j' \in [k]$ with $(i,j) \neq (i',j')$, the graph $H[M(i,j)]$ is connected for all $i, j \in [k]$, and contracting each set $M(i,j)$ to one vertex (i,j) results in $\tilde{H}$, see Figure 3.10. Furthermore, define $M(\tilde{Z}) = \bigcup_{(i,j)\in\tilde{Z}} M(i,j)$ for sets $\tilde{Z} \subseteq \tilde{V}_H$. Property (H1) implies that $M(\tilde{V}_H) = V_H$ and the following two properties.

(H5) The graph $H[M(i,j)]$ is a tree for all $i, j \in [k]$.

(H6) For all $i, j, i', j' \in [k]$ that satisfy $|i - i'| + |j - j'| = 1$, there is exactly one edge in H that joins a vertex in $M(i,j)$ to a vertex in $M(i',j')$.
For all $i, j, i', j' \in [k]$ with $(i,j) \neq (i',j')$ and $|i - i'| + |j - j'| \neq 1$, there is no edge in H that joins a vertex in $M(i,j)$ to a vertex in $M(i',j')$.

For $i, j \in [k-1]$, let $\tilde{A}_{i,j}$ be the 4-cycle $((i,j), (i+1,j), (i+1,j+1), (i,j+1))$ in the grid $\tilde{H}$. The small faces of $\tilde{H}$ in an arbitrary embedding of $\tilde{H}$ are exactly the faces bounded by the cycles $\tilde{A}_{i,j}$ with $i, j \in [k-1]$, see Figure 3.10. Properties (H5) and (H6) imply that for all $i, j \in [k-1]$ the subgraph of H induced by the vertices in $M(V(\tilde{A}_{i,j}))$ contains a unique cycle $A_{i,j}$ and this cycle bounds a small face of H. For $i, j \in [k-1]$, let $F_{i,j}$ be the set of vertices in V' that are on the cycle $A_{i,j}$ or embedded inside the cycle $A_{i,j}$ in the given embedding of G'. Furthermore, let A_{large} be the cycle in H that bounds the large face of H and define $C_k = M(\tilde{C}_k) \cap V(A_{\text{large}})$ as well as $R_k = M(\tilde{R}_k) \cap V(A_{\text{large}})$. For all $j \in [k-1]$, let $F_{k,j} = F_{k-1,j} \cap C_k$, for all $i \in [k-1]$, let $F_{i,k} = F_{i,k-1} \cap R_k$, and $F_{k,k} = F_{k-1,k-1} \cap C_k \cap R_k = F_{k-1,k-1} \cap M(k,k)$. For all

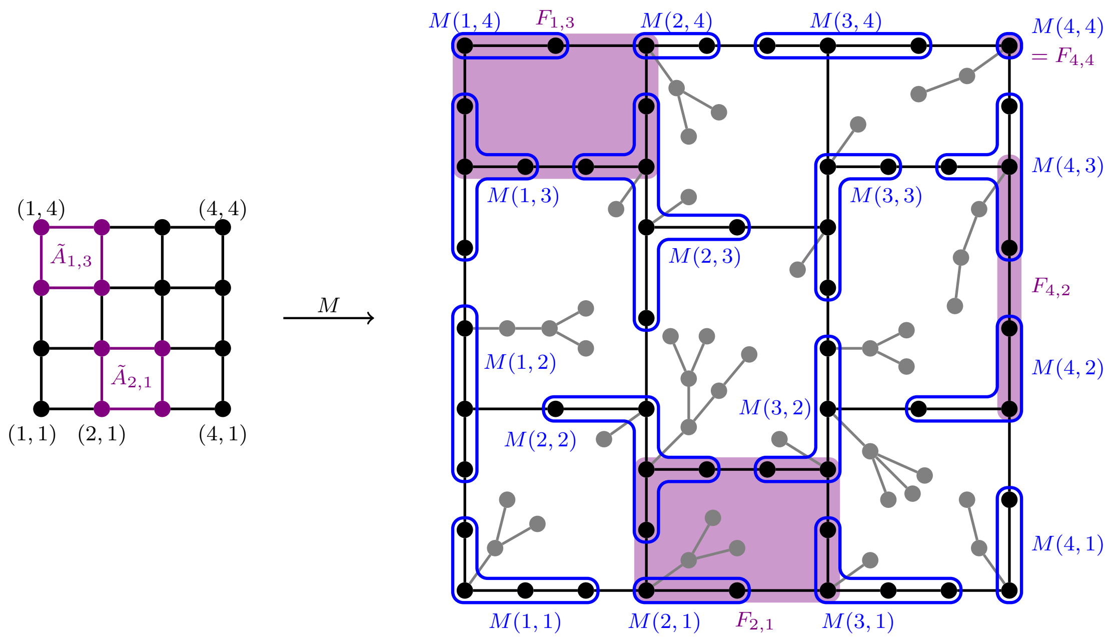


Figure 3.10: Notation used in the proof of Lemma 3.21. The $k \times k$ grid $\tilde{H}$ is drawn on the left and the graph G' is drawn on the right. The subgraph H is colored black. The vertices and edges, that are in G' but not in H, are colored gray. The cycles $\tilde{A}_{1,3}$ and $\tilde{A}_{2,1}$ in $\tilde{H}$ are colored purple. Their corresponding cycles $A_{1,3}$ and $A_{2,1}$ in H define the sets $F_{1,3}$ and $F_{2,1}$ in G', which are highlighted in purple. Even though Definition 3.18 and Lemma 3.21 require $k \geq 5$, we chose $k = 4$ to keep the example small.

remaining integers i, j, i. e., all pairs (i, j) with $i \notin [k]$ or $j \notin [k]$, define $F_{i,j} = \emptyset$. Then (H2) implies

$$|F_{i,j}| \leq \ell \qquad \text{for all } i, j \in \mathbb{N}_0. \tag{3.10}$$

It is easy to see that, for all $i, j, i', j' \in \mathbb{N}_0$ with $|i - i'| \geq 2$,

$$F_{i,j} \cap F_{i',j'} = \emptyset \tag{3.11}$$

and that, for all $i, j, j' \in [k]$,

$$\begin{aligned} M(i,j) &\subseteq F_{i-1,j-1} \cup F_{i,j-1} \cup F_{i,j} \cup F_{i-1,j}, \\ M(i,j) \cap F_{i,j} &\neq \emptyset, \\ M(i,j) \cap F_{i+1,j'} &= \emptyset. \end{aligned} \tag{3.12}$$

Now, the collection of clusters $\mathcal{X} = (X^i)_{i \in [k]}$ is defined by

$$X^i = \bigcup_{j \in [k]} (F_{i,j} \setminus F_{i+1,j})$$

for all $i \in [k]$, see Figure 3.11.

The aim is to show that G is $(\hat{\gamma}, \hat{k}, \hat{c})$-path-prosperous by showing that G' and $\mathcal{X}$ satisfy the properties in Definition 3.20. Clearly, $|V'| \geq (1 - \hat{\gamma})|V|$ holds for $\hat{\gamma} = \gamma$. Furthermore, (H1) and (H3) imply $\bigcup_{i \in [k]} \bigcup_{j \in [k]} F_{i,j} = V'$, which together with (3.11) shows that

$$V' = \dot{\bigcup_{i \in [k]}} X^i. \tag{3.13}$$

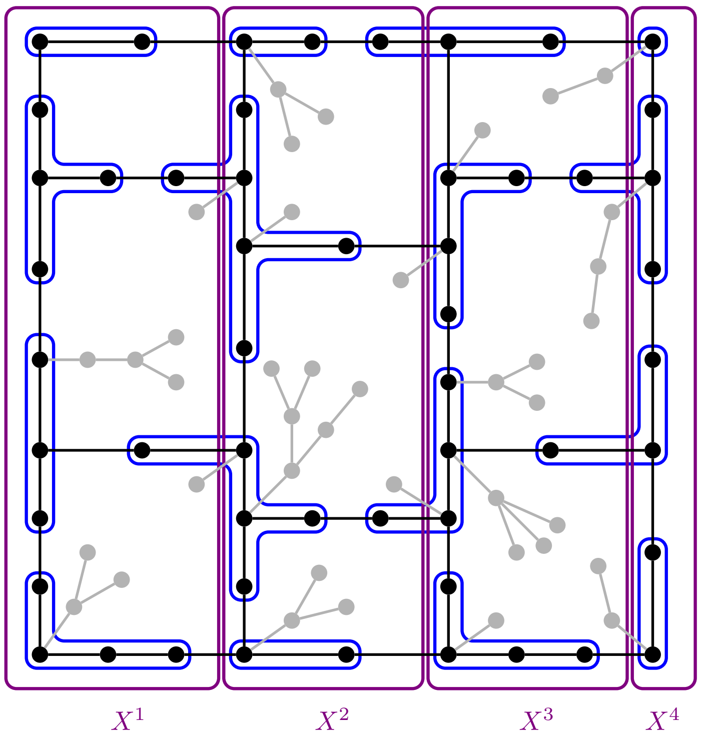


Figure 3.11: Definition of the sets $\mathcal{X} = (X^i)_{i\in[k]}$ used in the proof of Lemma 3.21. The set X^4 depends on some of the sets $M(i, j)$, which are indicated in blue. As in Figure 3.10, we chose $k = 4$ to keep the example small.

Therefore, (P1) is satisfied. Furthermore, (3.12) implies that, for every $i \in [k]$, the set X^i contains at least one vertex from $M(i, j)$ for every $j \in [k]$. As the sets $M(i, j)$ are pairwise disjoint for all $i, j \in [k]$, the set X^i contains at least k vertices for every $i \in [k]$. Hence, (P2) is satisfied with $\hat{k} = k$.

To show that (P3) is satisfied with $\hat{c} = \frac{1}{4\ell}$, choose $i_1, i_2 \in [\hat{k}]$ with $i_1 \le i_2$ arbitrarily, as well as $Z_1 \subseteq X^{i_1}$ and $Z_2 \subseteq X^{i_2}$ with $|Z_1| = |Z_2| = d$ for some $d \in [\hat{k}]$. We will show that there are at least $\lceil \frac{1}{2} d' \rceil$ edge-disjoint Z_1,Z_2-paths in G', where $d' := \lceil \frac{d}{2\ell} \rceil$.

Case 1: $\hat{k} = k < 8$. Note that in this case, the aim is to find *one* edge-disjoint Z_1,Z_2-path in G', because $d \le \hat{k}$ is required by (P3) and $\ell \ge 4$ as each face of H is bounded by a cycle of length at least 4. As G' is connected, there must be a path P in G' joining a vertex in Z_1 to a vertex in Z_2. If necessary, P can be shortened to contain exactly one vertex in Z_1 and exactly one vertex in Z_2 to obtain the desired Z_1,Z_2-path.

Case 2: $k \ge 8$. The idea of this case is to use Lemma 3.23, which ensures the existence of many vertex-disjoint paths in square grids. These paths can be mapped to paths joining two *glueing vertices* u_1 and u_2 in the subgraph H. To extend these paths such that each of them connects a vertex from Z_1 with a vertex from Z_2, paths from vertices in Z_1 and Z_2 to the glueing vertices u_1 and u_2, respectively, are considered and these paths only use vertices that are close to the glueing vertices. This will ensure that the Z_1,Z_2-paths are edge-disjoint in the end.

Recall that $i_1 \in [\hat{k}]$ with $Z \subseteq X^{i_1}$. For every $j \in [k]$, define $M(k + 1, j) = \emptyset$ and

$$N_j^1 = M(i_1, j) \cup M(i_1 + 1, j).$$

Clearly, N_j^1 and $N_{j'}^1$ are disjoint for distinct $j, j' \in [k]$. For every $j \in [k]$, choose an arbitrary $u_j^1 \in M(i_1, j)$. For every $j \in [k]$, and for every $v \in N_j^1$, the vertex u_j^1 is called the *glueing vertex* of v. Property (H4)

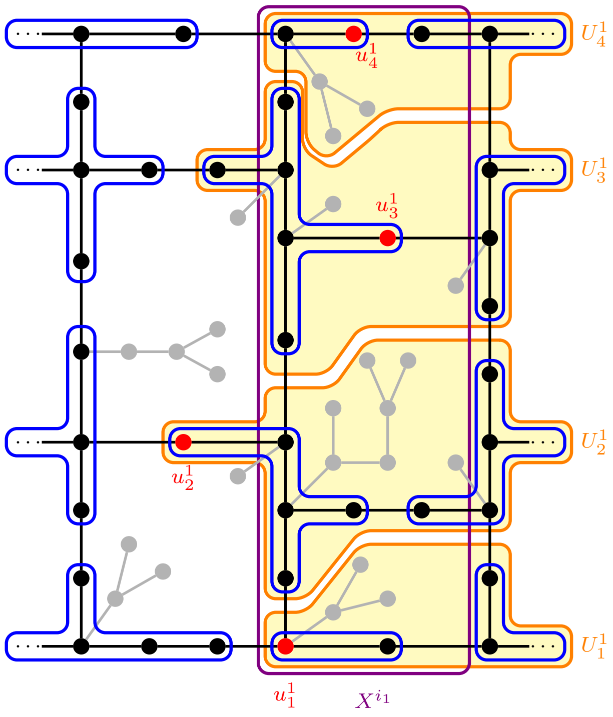


Figure 3.12: Definition of the sets U_j^1 for $j \in [k]$ in the proof of Lemma 3.21.

implies that for every $x \in V' \setminus V_H$ there is a unique vertex $v \in V_H$ that is closest to x in G'. For each vertex $x \in X^{i_1} \setminus V_H$, let v be this unique vertex in V_H that is closest to x in G' and note that there is some $j \in [k]$ such that $v \in N_j^1$. Now, let the glueing vertex of x be the glueing vertex of v. Then, every $x \in X^{i_1}$ has a unique glueing vertex u and G' contains an x,u-path that uses only vertices whose glueing vertex is u, and this x,u-path is unique due to (H4)-(H6). For every $j \in [k]$, denote by U_j^1 the set of vertices in V', whose glueing vertex is u_j^1, see Figure 3.12. Note that U_j^1 and $U_{j'}^1$ are disjoint for distinct $j, j' \in [k]$. By construction,

$$U_j^1 \subseteq F_{i_1,j-1} \cup F_{i_1,j} \cup M(i_1, j) \cup M(i_1 + 1, j)$$

for every $j \in [k]$ and $U_j^1 \cap X^{i_1} \subseteq F_{i_1,j-1} \cup F_{i_1,j}$ for every $j \in [k]$ by the definition of X^{i_1}. So, (3.10) implies that $|U_j^1 \cap Z_1| \leq 2\ell$ for every $j \in [k]$.

Let $J_1 = \{j \in [k]\colon U_j^1 \cap Z_1 \neq \emptyset\}$ and note that $|J_1| \geq \left\lceil \frac{|Z_1|}{2\ell} \right\rceil = d'$. For every $j \in J_1$, let x_j^1 be an arbitrary vertex in $U_j^1 \cap Z_1$ and denote by P_j^1 the unique x_j^1,u_j^1-path in G' that uses only vertices in U_j^1. Note that $V(P_j^1) \cap V_H \subseteq N_j^1$ for all $j \in J_1$. Define analogously N_j^2, u_j^2, and U_j^2 for all $j \in [k]$, J_2, as well as x_j^2 and P_j^2 for all $j \in J_2$ using the set $Z_2 \subseteq X^{i_2}$ instead of $Z_1 \subseteq X^{i_1}$. The aim is to find $\left\lceil \frac{1}{2} d' \right\rceil$ edge-disjoint paths from $\{x_j^1\colon j \in J_1\}$ to $\{x_j^2\colon j \in J_2\}$.

By construction, the set $U_{j_1}^1$ intersects the set $U_{j_2}^2$ if and only if $j_1 = j_2$ and $i_2 \in \{i_1, i_1 + 1\}$. To make these sets disjoint in the case that $i_2 = i_1$ or $i_2 = i_1 + 1$, do the following for every $j \in J_1 \cap J_2$. Join the vertices x_j^1 and x_j^2 by a path that uses only vertices in $U_j^1 \cup U_j^2$, delete j from J_1 and J_2, and decrease d' by one. Hence, in the following, we may assume that $U_{j_1}^1$ does not intersect $U_{j_2}^2$ for every $j_1 \in J_1$ and every $j_2 \in J_2$.

Let $\tilde{Z}_1 = \{(i_1, j)\colon j \in J_1\}$ and $\tilde{Z}_2 = \{(i_2, j)\colon j \in J_2\}$. As $k \geq 8$, $|\tilde{Z}_1| \geq d'$, and $|\tilde{Z}_2| \geq d'$, Lemma 3.23 implies that there are $\left\lceil \frac{1}{2} d' \right\rceil$ vertex-disjoint $\tilde{Z}_1$,$\tilde{Z}_2$-paths in the grid $\tilde{H}$, such that none of the paths uses a vertical edge in a column $\tilde{C}_i$ with $i \in I_f$ or a horizontal edge in a row $\tilde{R}_j$ with $j \in J_f$, where $I_f = \{i_1, i_1+1, i_2, i_2+1\} \cap [k]$ and $J_f = \{j \in [k]\colon\ \tilde{R}_j \cap (\tilde{Z}_1 \cup \tilde{Z}_2) = \emptyset\}$. Let $\tilde{\mathcal{P}}$ be the set of these $\tilde{Z}_1$,$\tilde{Z}_2$-paths in $\tilde{H}$. For every path $\tilde{P} \in \tilde{\mathcal{P}}$, with ends (i_1, j_1) and (i_2, j_2), let P be the $u^1_{j_1}$,$u^2_{j_2}$-path in H that uses only vertices in $M(V(\tilde{P}))$. Such a path always exists and is unique as $u^1_{j_1} \in M(i_1, j_1)$, $u^2_{j_2} \in M(i_2, j_2)$, and the subgraph of H induced by $M(V(\tilde{P}))$ is a tree due to (H5) and (H6). Let $\mathcal{P}$ be the set containing the path P for every path $\tilde{P} \in \tilde{\mathcal{P}}$. Due to the constraints on the paths in Lemma 3.23, it follows that, for every $j \in J_1$ and every $i \in I_f$, the vertex (i, j) can only be used in a path $\tilde{P} \in \tilde{\mathcal{P}}$ if (i, j) or $(i-1, j)$ is an end of $\tilde{P}$ and similarly for every $j \in J_2$. By the construction of $\mathcal{P}$ and because $U^1_{j_1} \cap U^2_{j_2} = \emptyset$ for every $j_1 \in J_1$ and every $j_2 \in J_2$ due to the previous assumption, every $u^1_{j_1}$,$u^2_{j_2}$-path in $\mathcal{P}$ uses neither a vertex in U^1_j for any $j \in J_1 \setminus \{j_1\}$ nor a vertex in U^2_j for any $j \in J_2 \setminus \{j_2\}$. Furthermore, note that none of the paths in $\mathcal{P}$ intersects with a previously constructed path in the case $i_2 = i_1$ or $i_2 = i_1 + 1$. Every $u^1_{j_1}$,$u^2_{j_2}$-path P in $\mathcal{P}$ can be extended to an $x^1_{j_1}$,$x^2_{j_2}$-walk by glueing P together with the path $P^1_{j_1}$ in $u^1_{j_1}$ and the path $P^2_{j_2}$ in $u^2_{j_2}$. Let $\mathcal{Q}$ be the set of these walks in G' and note that each such walk starts in Z_1 and ends in Z_2. Using that $V(P^1_j) \subseteq U^1_j$ for all $j \in J_1$ and that $V(P^2_j) \subseteq U^2_j$ for all $j \in J_2$, one can see that the walks in $\mathcal{Q}$ are pairwise vertex-disjoint. Shortening the walks to paths and further shortening paths that use several vertices in Z_1 or Z_2, if necessary, gives the desired $\lceil \frac{1}{2} d' \rceil$ edge-disjoint Z_1,Z_2-paths and shows that (P3) is satisfied. □

Observe that for deriving the properties (H5) and (H6) in the previous proof, it was only used that H is a minimal graph that contains a $k \times k$ grid as a minor. As these properties will be used again in Section 3.3, they are stated once more in the next proposition.

Proposition 3.24.
Let $k \geq 3$ and let $H = (V_H, E_H)$ be a minimal graph containing a $k \times k$ grid as a minor. Then, V_H can be partitioned into non-empty sets $M_{i,j}$ with $i, j \in [k]$ such that contracting $M_{i,j}$ to one vertex (i, j) results in the $k \times k$ grid. Every such partition of V_H satisfies the following.

a) For all $i, j \in [k]$, the graph $H[M_{i,j}]$ is a tree.

b) For all $i, j, i', j' \in [k]$ that satisfy $|i - i'| + |j - j'| = 1$, there is exactly one edge in H joining a vertex in $M_{i,j}$ to a vertex in $M_{i',j'}$.
For all $i, j, i', j' \in [k]$ with $(i, j) \neq (i', j')$ and $|i - i'| + |j - j'| \neq 1$, there is no edge in H joining a vertex in $M_{i,j}$ to a vertex in $M_{i',j'}$.

Consider a bisection (B, W) in a (γ, k, c)-path-prosperous graph G and let G' and $\mathcal{X}$ be as in Definition 3.20. Intuitively, there must be a cluster in $\mathcal{X}$ with many vertices in B as well as a cluster with many vertices in W, where many vertices refers to a linear fraction of the vertices in the cluster. Then, (P3) guarantees many edge-disjoint paths that start in B and end in W, which will now be used to derive the lower bound on the width of a minimum bisection in path-prosperous graphs from Lemma 3.22.

Proof of Lemma 3.22. Fix some $0 \leq \gamma < \frac{1}{2}$ and let $G = (V, E)$ be an arbitrary (γ, k, c)-path-prosperous graph with $|V|$ even. Then, there is a subgraph $G' = (V', E') \subseteq G$ and a collection of clusters $\mathcal{X} = (X^i)_{i \in I}$ such that the properties (P1)-(P3) in Definition 3.20 are satisfied and $|V'| \geq (1 - \gamma)\,|V|$, which is equivalent to

$$|V \setminus V'| \ \leq\ \gamma\,|V|\,. \tag{3.14}$$

Let (B, W) be an arbitrary bisection in G. The aim is to show that $e_G(B, W) \geq \left(\frac{1}{2} - \gamma\right) ck$, which then completes the proof of the lemma. Using $|B| = \frac{1}{2}|V|$, it follows that

$$|B \cap V'| \ \geq\ |B| - |V \setminus V'| \ \overset{(3.14)}{\geq}\ \left(\tfrac{1}{2} - \gamma\right)|V| \ \geq\ \left(\tfrac{1}{2} - \gamma\right)|V'|\,. \tag{3.15}$$

The fact that $(X^i)_{i\in I}$ is a partition of V' due to (P1) yields

$$\sum_{i\in I} |X^i \cap B| \;=\; |B \cap V'| \;\overset{(3.15)}{\geq}\; \left(\tfrac{1}{2}-\gamma\right)|V'| \;=\; \left(\tfrac{1}{2}-\gamma\right)\sum_{i\in I}|X^i|\,,$$

which implies that there exists an $i_B \in I$ with $|X^{i_B}\cap B| \;\geq\; \left(\frac{1}{2}-\gamma\right)|X^{i_B}|$. Analogously, one can argue that there exists an $i_W \in I$ with $|X^{i_W}\cap W| \;\geq\; \left(\frac{1}{2}-\gamma\right)|X^{i_W}|$. Using (P2), it follows that $|X^{i_B}\cap B| \geq \left(\frac{1}{2}-\gamma\right)k$ and $|X^{i_W}\cap W| \geq \left(\frac{1}{2}-\gamma\right)k$. Let $d := \min\{|X^{i_B}\cap B|, |X^{i_W}\cap W|, k\}$, which satisfies $\left(\frac{1}{2}-\gamma\right)k \leq d \leq k$. Now, choose $Z_B \subseteq X^{i_B}\cap B$ and $Z_W \subseteq X^{i_W}\cap W$ with $|Z_B| = |Z_W| = d$. There are at least cd edge-disjoint Z_B,Z_W-paths in G' by (P3) and, hence, also in G. Each of these paths has length at least one, i. e., it contains at least one edge, since $Z_B \cap Z_W \subseteq B \cap W = \emptyset$. Consequently, each path contains at least one cut edge, which is cut by (B, W), and $e_G(B, W) \geq cd \geq \left(\frac{1}{2}-\gamma\right)ck$, as we wanted to show. □

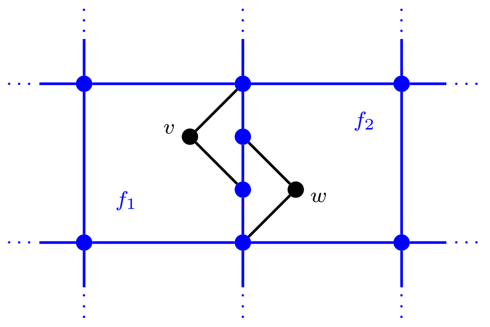
3.3 Investigating the Algorithmic Use of Grid-Homogeneous Graphs

Fix an arbitrary $0 \leq \gamma < \frac{1}{2}$, an integer $\ell \in \mathbb{N}$, an integer $\Delta_0 \in \mathbb{N}$, and a $c > 0$. Consider a family $\mathcal{G}$ of planar graphs where each graph $G \in \mathcal{G}$ on n vertices is (γ, k, ℓ)-grid-homogeneous with $k \geq c\sqrt{n}$ and satisfies $\Delta(G) \leq \Delta_0$, for example the graphs in Figure 3.5 on Page 59. As mentioned in Section 1.2.2, there is a constant c' such that the algorithm contained in Theorem 1.10 is a c'-approximation for the Minimum Bisection Problem restricted to the class $\mathcal{G}$. So the following questions are at hand: Given a bounded-degree planar graph, is there a polynomial-time algorithm that checks whether G is (γ, k, ℓ)-grid-homogeneous for certain values of γ, k, and ℓ? Given a bounded-degree planar graph, is there a polynomial-time algorithm that computes γ, k, and ℓ such that G is (γ, k, ℓ)-grid-homogeneous and the lower bound in Theorem 1.7 is maximized? This section studies such questions related to asking whether a graph is grid-homogeneous.

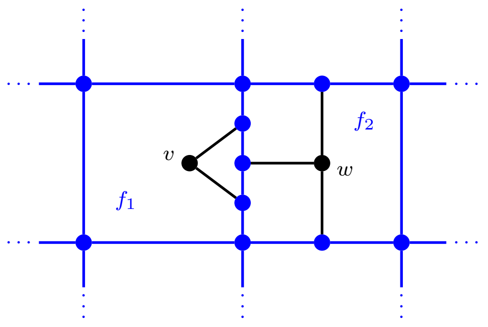
Section 3.3.1 gives an overview of the considered problems and states the hardness and approximability results derived here. The first problem, called the Homogeneous Embedding into a Square Grid Problem, or short the HEG Problem, focuses on the parameter ℓ of (γ, k, ℓ)-grid-homogeneous graphs. The second problem, called the Homogeneous Grid Minor Problem, or short the HGM Problem, focuses on choosing the subgraph H as in the definition of a (γ, k, ℓ)-grid-homogeneous graph G, i. e., a minimal graph containing a $k\times k$ grid as a minor that is, roughly speaking, spread homogeneously through G. All hardness results presented in Section 3.3.1 follow from reductions from a special version of 3-SAT, which is introduced in Section 3.3.2. Sections 3.3.3-3.3.5 present the proofs for the results stated in Section 3.3.1.

3.3.1 The HEG Problem and the HGM Problem

Consider a connected planar graph $G = (V, E)$ on n vertices that contains a $k\times k$ grid as a minor with $k \geq 5$. Let $H \subseteq G$ be a minimal graph that contains a $k\times k$ grid as a minor. Furthermore, consider an embedding of H in the plane, which can be extended to an embedding of G such that all properties in Definition 3.18 are satisfied for $G' = G$, except possibly condition (H2), which says that, for some fixed $\ell \in \mathbb{N}$, each face of the induced embedding of H contains at most ℓ vertices including the vertices on the boundary of the face. First, the problem of deciding whether (H2) can be satisfied is discussed under the assumption that $G = G'$ and H are given as input. This is made precise by stating the HEG Problem and its optimization version, the Min HEG Problem. There and in the remaining chapter, the following convention is used. When counting the vertices that are embedded in a face f of a plane graph, then the vertices on the boundary of f are counted as well.



a) The components $(\{v\}, \emptyset)$ and $(\{w\}, \emptyset)$ of $G - V(H)$ each can be embedded in the faces f_1 and f_2 of H but not both at the same time.



b) If the component $(\{v\}, \emptyset)$ is embedded in the face f_2 of H, then the embedding cannot be extended to an embedding of G without crossing edges.

Figure 3.13: Choosing where to embed the components of $G - V(H)$. The graph H is colored blue and not shown completely. The labels f_1 and f_2 refer to faces of the induced embedding of H.

Homogeneous Embedding Into a Square Grid (=HEG):
Input: $k \in \mathbb{N}$ with $k \geq 5$, $\ell \in \mathbb{N}$, a connected planar graph G, a plane subgraph $H \subseteq G$ such that H is a minimal graph containing the $k \times k$ grid as a minor.
Question: Can the embedding of H be extended to an embedding of G such that each small face of H contains at most ℓ vertices of G and no vertex of $V(G) \setminus V(H)$ is embedded in the large face of H?

Minimum Homogeneous Embedding Into a Square Grid (=Min HEG):
Input: $k \in \mathbb{N}$ with $k \geq 5$, a connected planar graph G, a plane subgraph $H \subseteq G$ such that H is a minimal graph containing the $k \times k$ grid as a minor.
Question: What is the smallest $\ell \in \mathbb{N}$ such that the embedding of H can be extended to an embedding of G where each small face of H contains at most ℓ vertices of G and no vertex of $V(G) \setminus V(H)$ is embedded in the large face of H? If there is no such $\ell \in \mathbb{N}$, return $\ell = \infty$.

Consider an instance (k, ℓ, G, H) of the HEG Problem. Roughly speaking, one needs to decide whether the vertices in $V(G) \setminus V(H)$ can be distributed homogeneously over the small faces of the plane graph H, where homogeneous is captured by the parameter ℓ. Similarly, for an instance (k, G, H) of the Min HEG Problem, one needs to determine the smallest integer ℓ such that the vertices in $V(G) \setminus V(H)$ can be spread homogeneously over the small faces of the plane graph H. Note that the input to both problems contains an embedding of the graph H and this is not a restriction due to Remark 3.15. Sometimes not only the optimal value of ℓ for the instance (k, G, H) of the Min HEG Problem is of interest, but also an embedding of the graph G with the properties required by the Min HEG Problem. The *corresponding embedding of* G for the instance (k, G, H) of the Min HEG Problem is an embedding of G, that is an extension of the embedding of H, such that no vertex in $V(G) \setminus V(H)$ is embedded in the large face of H and, for each small face f of H, there are at most ℓ vertices of G embedded in the face f, where ℓ denotes the optimal solution of the instance (k, G, H) of the Min HEG Problem.

Let (k, G, H) be an instance of the Min HEG Problem and consider a component $\tilde{G}$ of $G - V(H)$. Then, in every corresponding embedding of G, there is one face of H, where all vertices of $\tilde{G}$ are embedded. So, to find a solution for the instance (k, G, H) one needs to distribute the components of $G - V(H)$ to the faces of H. Choosing to embed one component $\tilde{G}$ in a face f of H can influence the possibilities to embed other components of $G - V(H)$. Figure 3.13a) shows an example where two components $\tilde{G}_1$ and $\tilde{G}_2$ of $G - V(H)$ can each be embedded in the faces f_1 and f_2 of H but not both at the same time. Figure 3.13b) shows that choosing to embed one component $\tilde{G}$ of $G - V(H)$ in a certain face of H might yield a plane embedding of $G[V(H) \cup V(\tilde{G})]$ that cannot be extended to a plane embedding of G.

Recall Definition 3.18, which defined a (γ, k, ℓ)-grid-homogeneous graph G, and denote by G' and H the subgraphs therein. If there is an edge $e \in E(G') \setminus E(H)$ such that $G'' = G' - e$ is connected, then the properties required in Definition 3.18 are also satisfied when using G'' instead of G'. If $G' - e$ is not connected for every $e \in E(G') \setminus V(H)$, then the problems displayed in Figure 3.13 cannot occur. More precisely, when constructing an embedding of G, then choosing to embed one component of $G' - V(H)$ in a face f of H does not affect the choices for embedding other components of $G' - V(H)$. Therefore, the following version of the HEG Problem is defined. Let Simplified Homogeneous Embedding Into a Square Grid, or short Simplified HEG, denote the restricted version of the HEG Problem, where the graph G has the additional property that $G - e$ is not connected for every $e \in E(G) \setminus E(H)$. Define analogously a restricted version of the Min HEG Problem, which is denoted by Simplified Minimum Homogeneous Embedding Into a Square Grid, or short Simplified Min HEG. Observe that the property that $G - e$ is not connected for every $e \in E(G) \setminus E(H)$ immediately implies that G is planar and that G has an embedding in the plane that is an extension of the embedding of the plane graph H, where no vertex in $V(G) \setminus V(H)$ is embedded in the large face of H.

The Simplified Min HEG Problem is similar to the following scheduling problem, which is examined by Lenstra, Shmoys, and Tardos in [LST90]. There, jobs need to be distributed to unrelated machines, which means that the processing time of a job depends on the machine to which it is assigned.

Scheduling on Unrelated Parallel Machines (=UPM Scheduling):
Input: number of jobs $n' \in \mathbb{N}$, number of machines $m' \in \mathbb{N}$, makespan $\ell' \in \mathbb{N}$, processing time $p'_{i,j} \in \mathbb{N}$ of job i on machine j for each $i \in [n']$ and $j \in [m']$.
Question: Is there a schedule such that all jobs are processed within ℓ' time units?

Min Scheduling on Unrelated Parallel Machines (=Min UPM Scheduling):
Input: number of jobs $n' \in \mathbb{N}$, number of machines $m' \in \mathbb{N}$, processing time $p'_{i,j} \in \mathbb{N}$ of job i on machine j for each $i \in [n']$ and $j \in [m']$.
Question: What is the smallest number $\ell' \in \mathbb{N}$ such that all jobs can be processed within ℓ' time units?

The UPM Scheduling Problem is NP-complete in the strong sense, since the more restricted version, where $p_{i,j} = p_{i,j'}$ for all $i \in [n']$ and all $j, j' \in [m']$ is required, is NP-complete in the strong sense, see Problem SS8 in [GJ79]. Next, it is argued that each instance of the Simplified HEG Problem can be converted into an equivalent instance of the UPM Scheduling Problem. Consider an instance $I = (k, \ell, G, H)$ of the Simplified HEG Problem. Let there be a machine for each small face of H, i. e., $m' = (k-1)^2$. Moreover, let there be a job for each component $\tilde{G}$ of $G - V(H)$ and for each face of H, i. e., n' is the number of components of $G - V(H)$ plus $(k-1)^2$. Define $\ell' = \ell$. Whenever a component $\tilde{G}$ of $G - V(H)$ can be embedded in the small face f of H let the processing time of the job corresponding to $\tilde{G}$ on the machine corresponding to f be the number of vertices of $\tilde{G}$ and otherwise define this processing time to be $\ell + 1$. To take care of the vertices on the boundary of each small face f of H, define the processing time of the job corresponding to f as the number of vertices on the boundary of f when scheduled on the machine corresponding to f and $\ell + 1$ when scheduled on any other machine. This defines an instance $I' = (n', m', \ell', (p'_{i,j}))$ of the UPM Scheduling Problem. Clearly, any embedding of G that shows that I is a yes-instance of the Simplified HEG Problem corresponds to a schedule of length at most ℓ' for the instance I' of the UPM Scheduling Problem. Observe that, in any feasible schedule of length at most ℓ' for the instance I', each job corresponding to a face f is assigned to the machine corresponding to f. Moreover, each job corresponding to a component $\tilde{G}$ of $G - V(H)$ is assigned to a machine that corresponds to a small face f of H such that $\tilde{G}$ can be embedded in f. Hence, the Simplified HEG Problem is a special version of the UPM Scheduling Problem.

On the positive side, it is known that there is a polynomial-time algorithm that finds a schedule whose length is at most twice the optimal length for each instance of the Min UPM Scheduling Problem, i. e., there is a 2-approximation for the Min UPM Scheduling Problem, see Theorem 2 in [LST90]. Recall that an algorithm is called an *α-approximation* for a minimization problem if it runs in polynomial time and it returns a solution that has cost at most α times the cost of an optimal solution. This implies the following corollary.

Corollary 3.25.
There is a 2*-approximation for the* Simplified Min HEG Problem*.*

On the negative side, it is NP-hard to approximate an optimal solution of the Min UPM Scheduling Problem within a factor less than $\frac{3}{2}$, meaning that, unless P = NP, there is no algorithm computing a schedule of length strictly less than $\frac{3}{2}$ the optimum length in polynomial time, see Theorem 5 and Corollary 2 in [LST90]. This does not yet imply that it is NP-hard to approximate an optimal solution of the Simplified Min HEG Problem within a factor smaller than $\frac{3}{2}$ and, indeed, it seems that the reduction used in [LST90] cannot be modified to a reduction for the HEG Problem. The reduction in the proof of Theorem 5 in [LST90] is from a problem called 3-Dimensional Matching, which considers a hypergraph on a vertex set $A \cup B \cup C$, where $|A| = |B| = |C| = n''$ and whose edge-set is a subset of $A \times B \times C$. The question is whether there are n'' edges whose union is $A \cup B \cup C$, i. e., each vertex in $A \cup B \cup C$ is in exactly one of the edges. From an instance of the 3-Dimensional Matching Problem, they create an instance of the UPM Scheduling Problem with processing times in $\{1, 2, 3\}$ and one asks whether there is a schedule of length 2. Hence, one can conclude that the UPM Scheduling Problem is NP-complete, even when all processing times are in $\{1, 2, 3\}$ and it is asked for a schedule of length 2. So the following result for the HEG Problem is not surprising.

Theorem 3.26.
The HEG Problem *is NP-hard, even for $\ell = 6$.*

The proof for the previous theorem is presented in Section 3.3.3. There, the Planar Monotone 3-SAT Problem is reduced to the HEG Problem, where all components of $G - V(H)$ have size one or two. It will follow that, even for $\ell = 6$, the HEG Problem is NP-hard. Here, the approach of a reduction from the 3-Dimensional Matching Problem is not considered, as it seems hard to deal with the geometric restrictions imposed by the grid in the HEG Problem compared to the UPM Scheduling Problem.

Observe also that, for $\ell = 5$, the Simplified HEG Problem can be solved in polynomial time in the following way. Let (k, ℓ, G, H) be an instance of the Simplified HEG Problem with $\ell = 5$. First, the algorithm checks whether each small face of the plane graph H contains at most five vertices on its boundary, which takes $\mathcal{O}(\|H\|)$ time by Lemma 2.26 as $\Delta(H) \leq 4$ by Proposition 3.16. If not, it returns no. So, from now on, assume that every small face of H has at most five vertices on its boundary. Note that each small face of H has at least four vertices on its boundary and that the small faces of H with four vertices on their boundary are the only faces of H in which vertices from $V(G) \setminus V(H)$ can be embedded, when trying to construct an embedding that shows that $(k, 5, G, H)$ is a yes-instance of the Simplified HEG Problem. Next, the algorithm computes a list L_{face} of all small faces of H that contain exactly four vertices on their boundary. Note that every edge in $G[V(H)]$ is also an edge of H and, hence, is already embedded. Deciding whether $(k, 5, G, H)$ is a yes-instance of the Simplified HEG Problem means to decide whether the vertices in $V(G) \setminus V(H)$ can be distributed to the faces in L_{face} such that each face f in L_{face} receives at most one of these vertices and, if a face f receives a vertex v, then v can be embedded in f. Next, the algorithm computes a list L_{comp} of all components of $G - V(H)$, which takes $\mathcal{O}(\|G\|)$ time by Lemma 2.25. If there is a component that contains two or more vertices, the algorithm returns no. So assume that every component of $G - V(H)$ contains exactly one vertex. Then, the algorithm checks which component in L_{comp} can be embedded in which face in L_{face}. Consider a component $\tilde{G}$ in L_{comp}, which is an isolated vertex v in $G - V(H)$ that has precisely one neighbor w in G as $(k, 5, G, H)$ is an instance of the Simplified HEG Problem. Hence, $\tilde{G}$ can be embedded in every face f that contains w on its boundary. Lemma 2.27 states that, for all vertices $w \in V(H)$, lists L_w with the faces that contain w on their boundary can be computed all together in $\mathcal{O}(\|H\|)$ time. The information, which component can be embedded in which face, is stored in a bipartite graph G_{emb}, whose vertices are the entries of L_{face} and L_{comp}. The graph G_{emb} contains an edge between a component $\tilde{G}$ in L_{comp} and a face f in L_{face} if and only if the component $\tilde{G}$ can be embedded in the face f. So, $(k, 5, G, H)$ is a yes-instance of the Simplified HEG Problem if and only if G_{emb} contains a matching that covers all vertices representing entries in L_{comp} or, equivalently, if a maximum matching in G_{emb} contains as many edges as entries of L_{comp}. The graph G_{emb} can be computed in $\mathcal{O}(\|G\|)$ time and the algorithm of Hopcroft and Karp [HK73] can be used to determine the cardinality of a maximum matching in G_{emb} in $\mathcal{O}(\|G_{\text{emb}}\|\sqrt{|V(G_{\text{emb}})|})$ time, which is polynomial in $\|G\|$.

Next, the minimization version of the HEG Problem, the Min HEG Problem is considered further. Clearly, the Min HEG Problem is NP-hard. As it is an optimization problem, it is natural to ask for approximations, i. e., is there an $\alpha > 1$ such that there is an α-approximation for the Min HEG Problem. When discussing the relation to the UPM Scheduling Problem, it was mentioned that the hardness results for the UPM Scheduling Problem do not immediately imply similar hardness results for the Min HEG Problem, but the following corollary can be derived directly from Theorem 3.26.

Corollary 3.27.
It is NP-hard to approximate the Min HEG Problem *within α for every $\alpha < \frac{7}{6}$.*

Proof. Fix an arbitrary $\alpha < \frac{7}{6}$. The idea is to show that approximating the Min HEG Problem within α is at least as hard as solving the HEG Problem for $\ell = 6$, which is NP-complete due to Theorem 3.26. Assume there is an α-approximation for the Min HEG Problem and let (k, ℓ, G, H) be an arbitrary instance of the HEG Problem with $\ell = 6$. Then (k, G, H) is an instance of the Min HEG Problem and applying the α-approximation returns a number ℓ_{app}. If $\ell_{\text{app}} \geq 7$, then the optimal value ℓ_{opt} of the instance (k, G, H) of the Min HEG Problem satisfies $\ell_{\text{opt}} \geq \frac{1}{\alpha}\ell_{\text{app}} > 6$. Therefore, (k, ℓ, G, H) is a no-instance of the HEG Problem. Otherwise, i. e., if $\ell_{\text{app}} < 7$, then $\ell_{\text{opt}} \leq \ell_{\text{app}}$ and $\ell_{\text{opt}} \leq 6$ as ℓ_{opt} is an integer. Therefore, (k, ℓ, G, H) is a yes-instance of the HEG Problem. Consequently, an α-approximation for the Min HEG Problem can be used to solve the HEG Problem. □

The previous corollary can be improved from $\frac{7}{6} \approx 1.166$ to 1.5 as stated by the next theorem. Its proof also relies on Theorem 3.26 but requires to look at the details of the proof of Theorem 3.26 and, hence, is presented later in Section 3.3.3.

Theorem 3.28.
It is NP-hard to approximate the Min HEG Problem *within α for every $\alpha < \frac{3}{2}$.*

Recall that, as discussed earlier, the UPM Scheduling Problem is NP-hard to approximate within α for every $\alpha < \frac{3}{2}$ as well, but this does not imply that the Min HEG Problem is NP-hard to approximate within α for every $\alpha < \frac{3}{2}$.

Next, some positive results on the approximability of the Min HEG Problem are presented. Recall that, as stated in Corollary 3.25, the algorithm by Lenstra et al. in [LST90] yields a 2-approximation for the Simplified Min HEG Problem. However, this algorithm relies on integer programming, relaxing the integrality constraints, and rounding techniques. Hence, it is complex to implement and it does not run in linear time. There are algorithms that compute an embedding of a planar graph in linear time, i. e., $\mathcal{O}(n)$ time for a graph G on n vertices, see Theorem 2.28. Consider an instance (k, G, H) of the Min HEG Problem. Since H is uniquely embeddable due to Remark 3.15, any embedding of G is an extension of the given embedding of the plane graph H. This is used to derive a linear-time 9-approximation as presented in the next lemma. An embedding of G *corresponds to an α-approximation* for the instance (k, G, H) of the Min HEG Problem if the embedding of G is an extension of the embedding of the plane graph H, where no vertex from $V(G) \setminus V(H)$ is embedded in the large face of H and each small face of H contains at most $\alpha\ell_{\text{opt}}$ vertices of G, where ℓ_{opt} denotes the optimal solution for the instance (k, G, H).

Lemma 3.29.
Let (k, G, H) be an instance of the Min HEG Problem *and denote by n the number of vertices of G. Any polynomial-time algorithm that computes an embedding of G, which is an extension of the embedding of the plane graph H such that no vertex of $V(G) \setminus V(H)$ is embedded in the large face of H, computes an embedding of G that corresponds to a 9-approximation for the* Min HEG Problem. *If no edge in $E(G) \setminus E(H)$ joins two vertices in H, then such an embedding of G can be computed in $\mathcal{O}(n)$ time.*

The proof of Lemma 3.29 is presented in Section 3.3.5, where it is also shown that the approximation ratio is tight. As argued earlier, when introducing the Simplified MIN HEG Problem, the restriction that G does not contain certain edges, which do not belong to H and can be removed without destroying the connectivity of G, is little. Observe that the restriction in the last lemma is weaker than requiring that (k, G, H) is an instance of the Simplified Min HEG Problem. Given an arbitrary instance (k, G, H) of the Min HEG Problem, it is easy to remove all edges in $E(G) \setminus E(H)$ that join two vertices in H in $\mathcal{O}(n)$ time, where n denotes the number of vertices of G. Indeed, recall that $V(G) = [n]$ and consider the following algorithm. First, the algorithm initializes a binary array A_H of length n with zeros, which takes $\mathcal{O}(n)$ time. Then, it set to one all entries of A_H that correspond to vertices in H by traversing H with a depth-first search to obtain a list of the vertices in H, which takes time proportional to $\|H\| \leq \|G\|$ by Lemma 2.25. Afterwards, for each vertex $v \in V(H)$, the algorithm traverses the adjacency list of v in G. Whenever an entry $u \in V(H)$ is discovered, the algorithm deletes u from the adjacency list of v in G if and only if u is not a neighbor of v in H, i. e., if and only if $\{u, v\}$ is not an edge of H. Observe that the array A_H can be used to check in constant time whether $u \in V(H)$ and, if so, it takes constant time to check whether u is a neighbor of v in H as $\Delta(H) \leq 4$ implies that the adjacency list of u in H can be traversed in constant time. Therefore, the traversals of the adjacency lists of G take $\mathcal{O}(\|G\|)$ time together. Consequently, the entire procedure takes $\mathcal{O}(n + \|G\|) = \mathcal{O}(n)$ time by Corollary 2.9.

When considering the simplified version of the Min HEG Problem and choosing the embedding of G in Lemma 3.29 a bit more carefully, then the following result is obtained.

Theorem 3.30.
There is a 5-approximation for the Simplified Min HEG Problem *that runs in $\mathcal{O}(n\sqrt{n})$ time when given an instance (k, G, H) where G is a graph on n vertices. Furthermore, without increasing the asymptotic running time, the algorithm can return a corresponding embedding of G.*

As Lemma 3.29, Theorem 3.30 is also proved in Section 3.3.5. There, it is also shown that the approximation ratio is tight with respect to the used construction. The reason for considering only instances (k, G, H) of the simplified version of the MIN HEG PROBLEM in Theorem 3.30, is that when extending the embedding of H to an embedding of G and choosing to embed one component of $G - V(H)$ in a small face of H, then this does not affect the possibilities where the other components of $G - V(H)$ can be embedded.

Recall Definition 3.18, which defines (γ, k, ℓ)-grid-homogeneous graphs, and consider a (γ, k, ℓ)-grid-homogeneous graph G with subgraphs G' and H as in Definition 3.18. The HEG PROBLEM focuses on the parameter ℓ or, more precisely, when G, G', and H are given to find a minimum ℓ and a corresponding embedding of G' such that the properties of Definition 3.18 are satisfied. Now, the problem of choosing the subgraph H is considered, i. e., finding an embedding of G' and a subgraph $H \subseteq G'$ that is a minimal graph containing a $k \times k$ grid as a minor and that is spread homogeneously through the graph G', which is now defined in detail.

HOMOGENEOUS GRID MINOR (=HGM):
Input: $k \in \mathbb{N}$ with $k \geq 5$, $\ell \in \mathbb{N}$, a connected planar graph G.
Question: Is there a subgraph $H \subseteq G$ such that H is a minimal graph containing a $k \times k$ grid as a minor and an embedding of G in the plane such that each small face of the induced embedding of H contains at most ℓ vertices of G and no vertex from $V(G) \setminus V(H)$ is embedded in the large face of H?

Consider an instance (k, ℓ, G) of the HGM PROBLEM. One part of the HGM PROBLEM is to decide whether the graph G contains a $k \times k$ grid as a minor. In general, it is NP-complete to decide whether an arbitrary graph contains a $k \times k$ grid as a minor for an arbitrary integer k that is part of the input [DB13] and it is open whether the problem remains NP-complete when restricted to planar graphs. For every constant $\varepsilon > 0$, there is an algorithm that, when given a planar graph G on n vertices, computes an integer k with $k \geq \frac{1}{3+\varepsilon}\,\mathrm{grid}(G)$ such that G contains a $k \times k$ grid as a minor in $\mathcal{O}(n^2 \log n)$ time [GT11].

As the HGM PROBLEM contains the HEG PROBLEM as a subproblem and the reduction used to prove Theorem 3.26 in Section 3.3.3 constructs a graph G that contains exactly one subgraph H that is a minimal graph containing a $k \times k$ grid as a minor, it follows that the HGM PROBLEM is NP-complete, even when only instances (k, ℓ, G) with $\ell = 6$ are considered. Therefore, we focus on instances (k, ℓ, G) of the HEG PROBLEM where G is uniquely embeddable. Recall that an embedding of G can be computed in $\mathcal{O}(n)$ time by Theorem 2.28. So, it is not needed to choose an embedding of G and once the subgraph H is chosen, there is nothing more to do as, for every vertex $v \in V(G) \setminus V(H)$, it is already determined in which face of H the vertex v is embedded. Thus, the reduction for showing that the HEG PROBLEM is NP-complete does not imply that this restricted version of the HGM PROBLEM is NP-complete.

Theorem 3.31.
The HGM PROBLEM *is NP-complete even when restricted to instances (k, ℓ, G) where G is uniquely embeddable.*

The proof of this theorem uses similar ideas as the proof of Theorem 3.26 and is presented in Section 3.3.4.

3.3.2 The SAT Problem and Selected Variants

Consider a set $\mathcal{U} = \{x_1, x_2, \ldots, x_n\}$ of n *Boolean variables*, i. e., $x_i \in \{\text{FALSE}, \text{TRUE}\}$ for all $i \in [n]$. A *Boolean formula* over $\mathcal{U}$ is built from the variables in $\mathcal{U}$, parenthesis, and the following operators, where x and y denote Boolean variables:

- *conjunction* $\wedge$, that is, $x \wedge y$ is true if x and y are both true,
- *disjunction* $\vee$, that is, $x \vee y$ is true if at least one of x and y is true, and
- *negation* $\bar{x}$, that is, $\bar{x}$ is true if x is false and vice versa.

A Boolean formula ϕ over $\mathcal{U}$ is said to be *satisfiable* if there is an assignment of True and False to the variables in $\mathcal{U}$ such that ϕ evaluates to True. For example,

$$\phi_1 := (x_1 \vee \bar{x}_1 \vee x_2 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_3)$$

is a Boolean formula over the set $\{x_1, x_2, x_3\}$. When setting $x_1 = \textsc{True}$, $x_2 = \textsc{False}$, and $x_3 = \textsc{False}$, then

$$\begin{aligned}\phi_1 &= (\textsc{True} \vee \textsc{False} \vee \textsc{False} \vee \textsc{True}) \ \wedge \ (\textsc{False} \vee \textsc{False} \vee \textsc{False}) \ \wedge \ (\textsc{True}) \\ &= \textsc{True} \wedge \textsc{False} \wedge \textsc{True} \ = \ \textsc{False}.\end{aligned}$$

Setting $x_1 = \textsc{True}$, $x_2 = \textsc{True}$, and $x_3 = \textsc{False}$ gives

$$\begin{aligned}\phi_1 &= (\textsc{True} \vee \textsc{False} \vee \textsc{True} \vee \textsc{False}) \ \wedge \ (\textsc{False} \vee \textsc{True} \vee \textsc{False}) \ \wedge \ (\textsc{True}) \\ &= \textsc{True} \wedge \textsc{True} \wedge \textsc{True} \ = \ \textsc{True},\end{aligned}$$

which shows that ϕ_1 is satisfiable. The famous NP-complete SAT Problem is the following.

SAT Problem:
Input: a Boolean formula ϕ.
Question: Is ϕ satisfiable?

Theorem 3.32 (Cook 1971, see Problem LO1 in [GJ79]).
The SAT Problem *is NP-complete.*

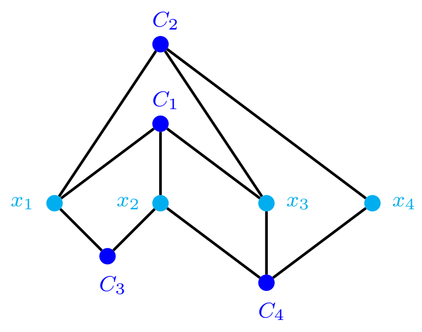
A *literal* is a variable or the negation of a variable. A *clause* is a disjunction of literals or a single literal. A Boolean formula is said to be in *conjunctive normal form* if it is a conjunction of clauses. For example, the Boolean formula ϕ_1 is in conjunctive normal form. It has three clauses, namely $x_1 \vee \bar{x}_1 \vee x_2 \vee \bar{x}_2$, $\bar{x}_1 \vee x_2 \vee x_3$, and $\bar{x}_3$. Here, only Boolean formulas in conjunctive normal form are considered. Note that a Boolean formula ϕ in conjunctive normal form is completely described by its set of variables $\mathcal{U}$ and its set of clauses $\mathcal{C}$. Therefore, we slightly abuse notation and write $\phi = (\mathcal{U}, \mathcal{C})$ in the following. A Boolean formula $\phi = (\mathcal{U}, \mathcal{C})$ in conjunctive normal form is a *3-SAT-formula* if each clause in $\mathcal{C}$ uses at most three literals. The Boolean formula ϕ_1 is not a 3-SAT formula as the first clause uses four literals. The Boolean formula

$$\phi_2 := (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee x_4) \wedge (x_1 \vee x_2) \wedge (x_2 \vee \bar{x}_3 \vee \bar{x}_4)$$

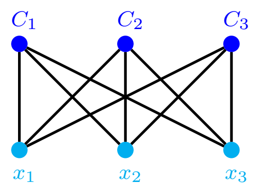
is a 3-SAT-formula. The SAT Problem remains NP-complete when restricted to 3-SAT-formulas.

3-SAT Problem:
Input: a 3-SAT-formula $\phi = (\mathcal{U}, \mathcal{C})$.
Question: Is ϕ satisfiable?

Theorem 3.33 (Karp 1972, see Problem L02 in [GJ79]).
The 3-SAT Problem *is NP-complete.*



a) Graph G_{ϕ_2}.



b) Graph G_{ϕ_4}.

Figure 3.14: Examples of planar and non-planar 3-SAT formulas. Vertices representing variables are colored light blue, vertices representing clauses are colored dark blue.

Variables and negations of variables are called *positive literals* and *negative literals*, respectively. Similarly, a clause is *positive* if it only contains positive literals and it is *negative* if it only contains negative literals. A clause that is either positive or negative is called *monotone* and a Boolean formula in conjunctive normal form is called *monotone* if each of its clauses is monotone. Neither ϕ_1 nor ϕ_2 is monotone, because they both contain the clause $\bar{x}_1 \vee x_2 \vee x_3$, which is not monotone. The formula

$$\phi_3 = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (x_3 \vee x_4 \vee x_5).$$

is an example of a monotone formula with two positive clauses and one negative clause. The SAT Problem and the 3-SAT Problem both remain NP-complete when restricted to monotone formulas.

Monotone SAT Problem:
Input: a SAT-formula ϕ in conjunctive normal form such that ϕ is monotone.
Question: Is ϕ satisfiable?

Monotone 3-SAT Problem:
Input: a monotone 3-SAT-formula $\phi = (\mathcal{U}, \mathcal{C})$.
Question: Is ϕ satisfiable?

Theorem 3.34 (Gold 1978 [Gol78], see Problem LO2 in [GJ79]).
The Monotone SAT Problem *and the* Monotone 3-SAT Problem *are NP-complete.*

Next, the concept of planarity is introduced. Let $\phi = (\mathcal{U}, \mathcal{C})$ be a Boolean formula in conjunctive normal form. Consider the bipartite graph G_ϕ on the color classes $\mathcal{U}$ and $\mathcal{C}$, where two vertices $x \in \mathcal{U}$ and $C \in \mathcal{C}$ are adjacent if and only if the clause C uses x or its negation $\bar{x}$. If G_ϕ is a planar graph, then ϕ is called *planar*. Figure 3.14a) shows that ϕ_2 is planar. Not every 3-SAT formula is planar, as for example the associated graph of

$$\phi_4 = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x}_3)$$

is isomorphic to the complete bipartite graph $K_{3,3}$, see Figure 3.14b), and $K_{3,3}$ is known to be non-planar, see Corollary 4.2.11 in [Die12]. When the 3-SAT Problem is restricted to planar formulas, it remains NP-complete.

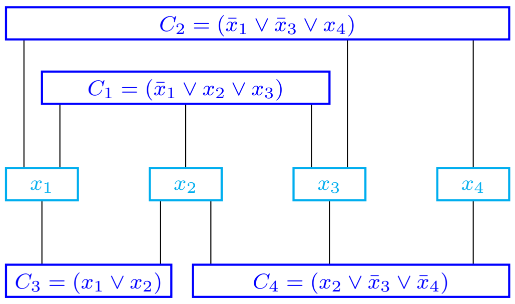

$C_2 = (\bar{x}_1 \vee \bar{x}_3 \vee x_4)$
$C_1 = (\bar{x}_1 \vee x_2 \vee x_3)$
x_1
x_2
x_3
x_4
$C_3 = (x_1 \vee x_2)$
$C_4 = (x_2 \vee \bar{x}_3 \vee \bar{x}_4)$

Figure 3.15: A rectilinear representation of ϕ_2. Similarly to Figure 3.14, rectangles representing variables are colored light blue and rectangles representing clauses are colored dark blue.

Planar 3-SAT Problem:
Input: a planar 3-SAT formula $\phi = (\mathcal{U}, \mathcal{C})$.
Question: Is ϕ satisfiable?

Theorem 3.35 (Lichtenstein 1982 [Lic82], or see Problem LO1 in [GJ79]).
The Planar 3-SAT Problem *is NP-complete.*

Consider a Boolean formula $\phi = (\mathcal{U}, \mathcal{C})$ in conjunctive normal form and a drawing, that represents ϕ, with the following properties: Each clause in $\mathcal{C}$ and each variable in $\mathcal{U}$ are represented by a rectangle, whose sides are horizontal and vertical lines. All rectangles corresponding to variables in $\mathcal{U}$ are drawn on a horizontal line. A rectangle corresponding to a variable $x \in \mathcal{U}$ is joined to a rectangle corresponding to a clause $C \in \mathcal{C}$ with a vertical line segment if and only if C uses the literal x or $\bar{x}$. None of these vertical line segments intersects a rectangle and only the rectangles corresponding to C and x are touched. A drawing with these properties is called a *rectilinear representation* of ϕ. See also the example in Figure 3.15.

In [KR92], Knuth and Raghunathan argue that it follows from the reduction in [Lic82] that every planar 3-SAT formula has a rectilinear representation, where planar is defined as in [Lic82]. Consider a Boolean formula $\phi = (\mathcal{U}, \mathcal{C})$ with $\mathcal{U} = \{x_1, x_2, \dots, x_n\}$ and the graph G_ϕ as defined above. In [Lic82], the graph G'_ϕ that is obtained from G_ϕ by adding the edges $\{x_i, x_{i+1}\}$ for all $i \in [n-1]$ and the edge $\{x_n, x_1\}$ is considered, and the Boolean formula ϕ is called planar if and only if G'_ϕ is planar. Note that this definition is more restrictive than the one above. It is not hard to see that when the graph G'_ϕ is planar, then there is a rectilinear representation of ϕ.

Next, the properties of rectilinear representations and monotone formulas are combined. Consider a planar 3-SAT-formula ϕ that is monotone. A *monotone rectilinear representation* of ϕ is a rectilinear

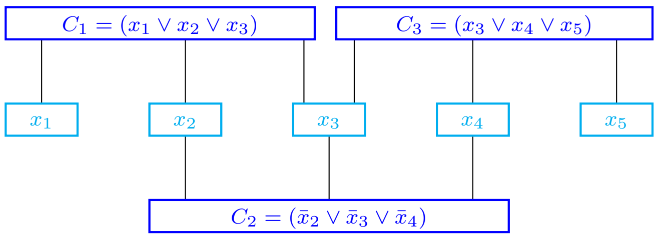

$C_1 = (x_1 \vee x_2 \vee x_3)$
$C_3 = (x_3 \vee x_4 \vee x_5)$
x_1
x_2
x_3
x_4
x_5
$C_2 = (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4)$

Figure 3.16: A monotone rectilinear representation of ϕ_3.

representation of ϕ such that all positive clauses are drawn above the variables and all negative clauses are drawn below the variables, where each variable and each clause were identified with the rectangle representing it. Figure 3.16 shows a monotone rectilinear representation of ϕ_3. Observe that not every planar and monotone Boolean formula automatically allows a monotone rectilinear representation, even if the more restricted version of planarity as in [Lic82] is required. For example,

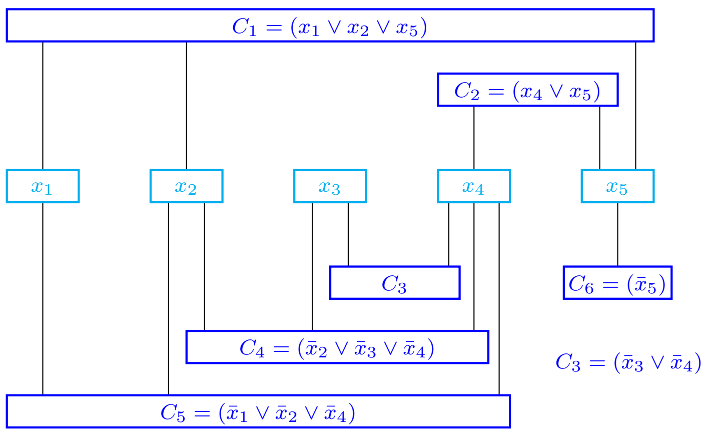
$$\phi_5 = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_3 \vee x_4 \vee x_5)$$

is monotone and planar, even when the edges $\{x_i, x_{i+1}\}$ for $i \in [4]$ and $\{x_5, x_1\}$ are added. Note that G_{ϕ_3} and G_{ϕ_5} are identical, but the requirements for a monotone rectilinear representation are different as the second clause of ϕ_3 is negative and the second clause of ϕ_5 is positive. So the second clause of ϕ_3 and ϕ_5 need to be drawn below and above the variables, respectively. It is not hard to see that ϕ_5 does not allow a monotone rectilinear representation. The 3-SAT Problem remains NP-complete when restricted to instances allowing a monotone rectilinear representation.

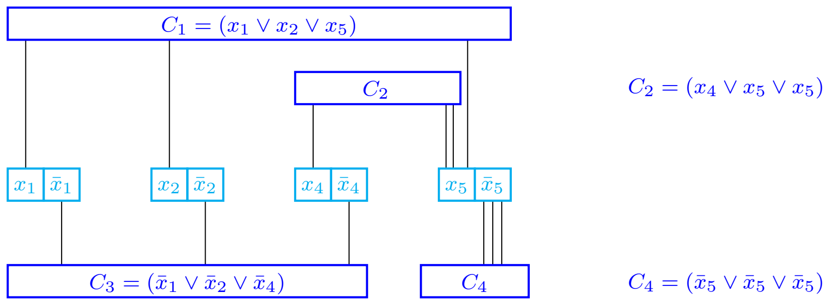
Planar Monotone 3-SAT Problem:
Input: a monotone rectilinear representation of a 3-SAT formula $\phi = (\mathcal{U}, \mathcal{C})$.
Question: Is ϕ satisfiable?

Theorem 3.36 (de Berg and Khosravi 2010 [BK10]).
The Planar Monotone 3-SAT Problem *is NP-complete.*

In the remaining section, when a monotone rectilinear representation of a 3-SAT formula is considered, clauses and variables are identified with the rectangles representing them in the considered monotone rectilinear representation. The reductions in Section 3.3.3 and Section 3.3.4 are from the Planar Monotone 3-SAT Problem and use the following notation. Consider an instance of the Planar Monotone 3-SAT Problem, i. e., a monotone rectilinear representation $\mathcal{R}$ of a 3-SAT formula $\phi = (\mathcal{U}, \mathcal{C})$. For $x \in \mathcal{U}$, denote by $\mathcal{L}^+(x)$ and $\mathcal{L}^-(x)$ the set of vertical lines in $\mathcal{R}$ that join x to a positive clause in $\mathcal{C}$ and a negative clause in $\mathcal{C}$, respectively. Moreover, for each $x \in \mathcal{U}$, define $\deg^+(x) := |\mathcal{L}^+(x)|$ as well as $\deg^-(x) := |\mathcal{L}^-(x)|$, i. e., the number of times that x appears as a positive and as a negative literal in ϕ, respectively. If there are two lines joining a variable x to a clause C, i. e., the clause C contains the variable x twice, then C contributes 2 to $\deg^+(x)$ or $\deg^-(x)$ depending on whether C is a positive or a negative clause. Due to technical reasons, some further assumptions on ϕ and $\mathcal{R}$ are needed and discussed now. First, one may assume that, for each $x \in \mathcal{U}$, there is a clause in which x appears as a positive literal and there is a clause in which x appears as a negative literal, without loss of generality. Indeed, assume that there is a variable $x \in \mathcal{U}$ such that x appears only as a positive literal in ϕ. The case when x appears only as a negative literal in ϕ is analogous. Let ϕ' be the formula obtained from ϕ by removing all clauses that contain x. It is easy to see that ϕ' is satisfiable if and only if ϕ is satisfiable as every satisfying assignment of ϕ' can be extended to a satisfying assignment of ϕ by setting x to True. Furthermore, ϕ' is monotone and planar, and a monotone rectilinear representation of ϕ' can be easily obtained from $\mathcal{R}$. Second, one may assume that every clause in $\mathcal{C}$ contains exactly three literals, as otherwise literals can be repeated. For example if there is a clause $C = (\bar{x})$ in $\mathcal{C}$, it can be replaced by $C' = (\bar{x} \vee \bar{x} \vee \bar{x})$ and two extra vertical lines joining x and C are added in $\mathcal{R}$. So, one may also assume that, in $\mathcal{R}$, each clause touches exactly three vertical lines that join the clause to variables. Third, one may assume that, in $\mathcal{R}$, each variable $x \in \mathcal{U}$ is split into a positive and a negative half, where the positive half is drawn on the left and the negative half is drawn on the right, such that all vertical lines in $\mathcal{L}^+(x)$ touch x at the positive


$C_1 = (x_1 \vee x_2 \vee x_5)$
$C_2 = (x_4 \vee x_5)$
x_1
x_2
x_3
x_4
x_5
C_3
$C_6 = (\bar{x}_5)$
$C_4 = (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4)$
$C_3 = (\bar{x}_3 \vee \bar{x}_4)$
$C_5 = (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_4)$

a) Drawing of ϕ_6.


$C_1 = (x_1 \vee x_2 \vee x_5)$
C_2
$C_2 = (x_4 \vee x_5 \vee x_5)$
x_1 $\bar{x}_1$
x_2 $\bar{x}_2$
x_4 $\bar{x}_4$
x_5 $\bar{x}_5$
$C_3 = (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_4)$
C_4
$C_4 = (\bar{x}_5 \vee \bar{x}_5 \vee \bar{x}_5)$

b) Drawing of ϕ_6''.

Figure 3.17: Modifications on the monotone rectilinear representation to satisfy the extra assumptions.

half and all vertical lines in $\mathcal{L}^-(x)$ touch x at the negative half. For an example, consider

$$\phi_6 := (x_1 \vee x_2 \vee x_5) \wedge (x_4 \vee x_5) \wedge (\bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_4) \wedge (\bar{x}_5).$$

The Boolean formula ϕ_6 contains the variable x_3 only as a negative literal, so the problem of deciding whether ϕ_6 is satisfiable can be simplified to deciding whether

$$\phi_6' := (x_1 \vee x_2 \vee x_5) \wedge (x_4 \vee x_5) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_4) \wedge (\bar{x}_5)$$

is satisfiable. To meet the requirement that each clause contains exactly three literals, let

$$\phi_6'' := (x_1 \vee x_2 \vee x_5) \wedge (x_4 \vee x_5 \vee x_5) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_4) \wedge (\bar{x}_5 \vee \bar{x}_5 \vee \bar{x}_5),$$

which is satisfiable if and only if ϕ_6' is satisfiable. Figure 3.17 shows a monotone rectilinear representation of ϕ_6 as well as a monotone rectilinear representation of ϕ_6'' that meets the extra requirement on dividing each variable into a positive and a negative half.

3.3.3 Proof of Hardness Results for the HEG Problem

Here, Theorem 3.26 is derived. First, the following weaker version is proved and then it is discussed how to modify the reduction to obtain Theorem 3.26.

Theorem 3.37.
The HEG PROBLEM *for $\ell \geq 10$ is NP-complete.*

Idea for the Reduction

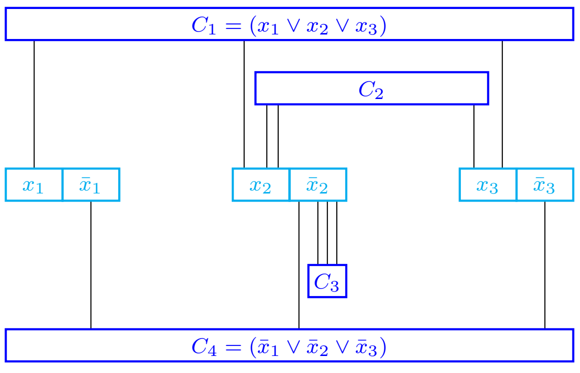
The proof of Theorem 3.37 relies on a reduction from the PLANAR MONOTONE 3-SAT PROBLEM. This means that, for any instance $I' = (\phi = (\mathcal{U}, \mathcal{C}))$ of the PLANAR MONOTONE 3-SAT PROBLEM, an instance $I = (k, \ell, G, H)$ of the HEG PROBLEM is constructed, such that I' is a yes-instance of the PLANAR MONOTONE 3-SAT PROBLEM if and only if I is a yes-instance of the HEG PROBLEM. Furthermore, the instance I needs to be computable from I' in time polynomial in the size of I'. The reduction shows that the HEG PROBLEM is at least as hard as the PLANAR MONOTONE 3-SAT PROBLEM, as the former one can be used to solve the latter one. In particular, if there was a polynomial-time algorithm solving the HEG PROBLEM, the reduction shows that there would also be a polynomial-time algorithm for the PLANAR MONOTONE 3-SAT PROBLEM.

The main idea of the reduction is that, if a 3-SAT formula ϕ is satisfied, then each of its clauses contains at least one literal that is TRUE, or, equivalently, each clause contains at most two literals that are FALSE, as each clause contains exactly three literals by assumption. In the following proof, a graph G_ϕ is constructed from a subdivision of a square grid H_ϕ. For each clause C of ϕ, a face f_C of H_ϕ will be specified and three little graphs, that are related to the literals used in C, will be attached to its boundary. The parameter ℓ and the subgraphs will be chosen such that at most two of the subgraphs can be embedded inside f_C if the embedding of G_ϕ has the properties required by the HEG PROBLEM. This corresponds to the idea that the clause C contains at most two literals that are FALSE, when a satisfying assignment of ϕ is considered. So, in the following proof, the little graphs added to H_ϕ can be interpreted as the FALSE of a truth-assignment. Furthermore, for each variable x, a little graph $\tilde{G}$ is added to H_ϕ and two faces f_x^+ and f_x^- of H_ϕ, called the positive and the negative face of x, are specified, such that $\tilde{G}$ can only be embedded in f_x^+ or f_x^-. The rectilinear representation of ϕ is used to join the faces f_x^- and f_x^+ of each variable x to the corresponding faces that represent a clause that contains x. Figure 3.18 gives an idea for this using the example

$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_2 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3).$$

Figure 3.18a) displays a monotone rectilinear representation of ϕ and Figure 3.18b) shows the graph H_ϕ with some faces colored according to the monotone rectilinear representation in Figure 3.18a). To obtain G_ϕ from H_ϕ, some little graphs are attached to the boundaries of these colored faces, such that each of these little graphs can only be embedded in a colored face when extending the embedding of H_ϕ to an embedding of G_ϕ. Therefore, the embedding of all other faces of G_ϕ is completely determined by the embedding of H_ϕ.

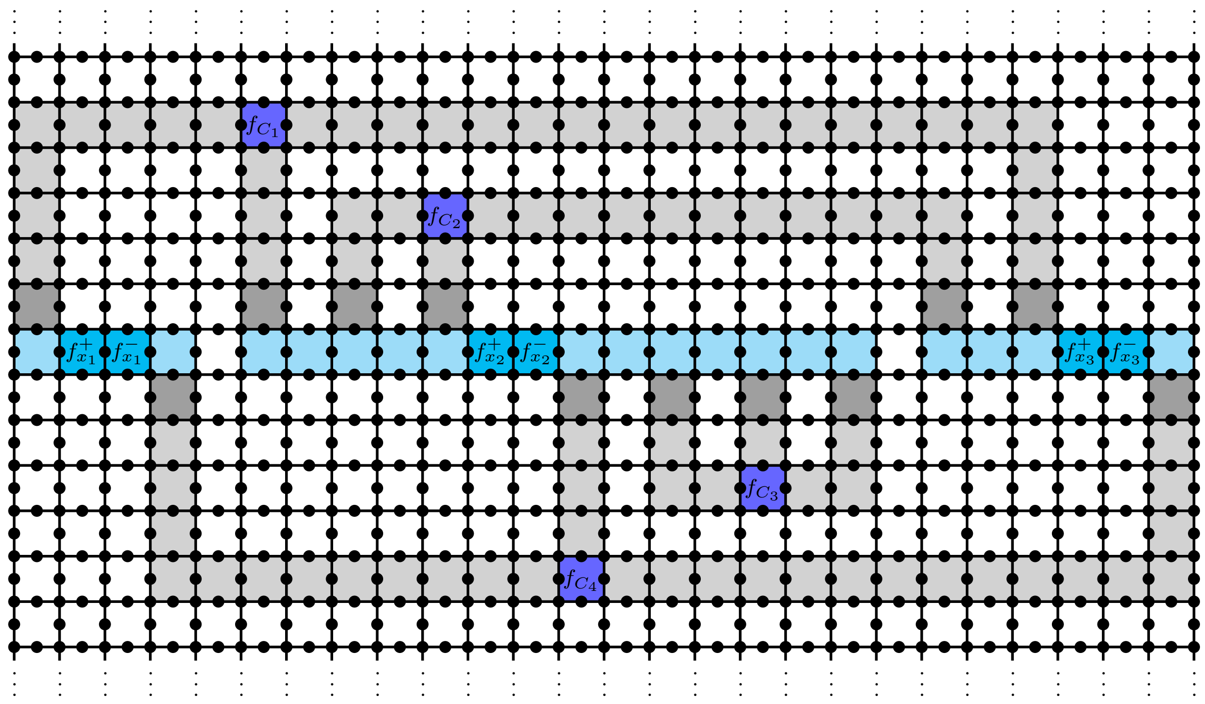
Consider a variable x and let $\tilde{G}$ be the subgraph added to H_ϕ due to x. When $\tilde{G}$ is embedded in the positive face f_x^+ we think of x sending the value FALSE to all positive clauses that contain x. Similarly, when $\tilde{G}$ is embedded in the negative face f_x^- we think of x sending the value FALSE to all negative clauses that contain $\bar{x}$. More precisely, consider an embedding of G_ϕ with the properties required by the HEG PROBLEM. If $\tilde{G}$ is embedded in f_x^+ and C is a positive clause containing x, meaning that the assignment $x =$ FALSE is considered, which does not help to satisfy the clause C, then there is a subgraph attached to the boundary of f_C that needs to be embedded in f_C. Furthermore, if $\tilde{G}$ is embedded in f_x^- and C is a negative clause containing $\bar{x}$, meaning that the assignment $x =$ TRUE is considered, which does not help to satisfy the clause C, then there is a subgraph attached to the boundary of f_C that needs to be embedded in f_C.



$C_1 = (x_1 \lor x_2 \lor x_3)$
C_2
x_1
$\bar{x}_1$
x_2
$\bar{x}_2$
x_3
$\bar{x}_3$
C_3
$C_4 = (\bar{x}_1 \lor \bar{x}_2 \lor \bar{x}_3)$

$C_2 = (x_2 \lor x_2 \lor x_3)$ $C_3 = (\bar{x}_2 \lor \bar{x}_2 \lor \bar{x}_2)$

a) Monotone rectilinear representation of ϕ.



b) The graph H_ϕ. To obtain the graph G_ϕ from H_ϕ, some little graphs are added to the boundaries of the colored faces.

Figure 3.18: Idea for the reduction to prove Theorem 3.37.

Notation

Consider an instance of the Planar Monotone 3-SAT Problem, i. e., a monotone rectilinear representation $\mathcal{R}$ of a 3-SAT formula $\phi = (\mathcal{U}, \mathcal{C})$. Define $n := |\mathcal{U}|$ and $m := |\mathcal{C}|$. Recall that the rectangles in $\mathcal{R}$ are identified with the corresponding variables and clauses in $\mathcal{C} \cup \mathcal{U}$. As discussed in the end of Section 3.3.2, we may assume without loss of generality that each variable in $\mathcal{U}$ is used at least once as a positive literal and at least once as a negative literal, and that each clause in $\mathcal{C}$ is a disjunction of three not necessarily distinct literals. Furthermore, in $\mathcal{R}$, each variable x can be split into a positive and a negative half, where the positive half is drawn on the left and the negative half is drawn on the right, such that all vertical lines in $\mathcal{L}^+(x)$ touch x at the positive half and all vertical lines in $\mathcal{L}^-(x)$ touch x at the negative half.

Set $k := 6m + n$ and let H_ϕ be the graph obtained from the $k \times k$ grid $\tilde{G}_k$ by subdividing each edge once. Recall that the vertex set of $\tilde{G}_k$ is $\{(i,j)\colon\ i \in [k], j \in [k]\}$. A vertex v in H_ϕ that subdivides an edge e of $\tilde{G}_k$ is called a *subdivision vertex* and is denoted by v_e. Clearly, H_ϕ is a minimal graph containing a $k \times k$ grid as a minor. Fix an embedding of H_ϕ in the plane and recall that H_ϕ is uniquely embeddable due to Remark 3.15. For the figures throughout the proof, the embedding of $\tilde{G}_k$ with the following properties is considered. For $i, j \in [k]$, the vertex (i, j) is embedded at the point (i, j) in a coordinate system whose horizontal axis refers to the first coordinate and whose vertical axis refers to the second coordinate. So, $(1, 1)$ is drawn in the bottom left corner and (k, k) is drawn in the top right corner. For each small face f of H_ϕ, there are integers $i, j \in [k-1]$ such that f is bounded by a cycle on the vertices $(i, j), (i+1, j), (i+1, j+1), (i, j+1)$ and the four subdivision vertices between them. Then,

- $v_{\{(i,j),(i+1,j)\}}$ is called the *bottom subdivision vertex* of f,
- $v_{\{(i+1,j),(i+1,j+1)\}}$ is called the *right subdivision vertex* of f,
- $v_{\{(i+1,j+1),(i,j+1)\}}$ is called the *top subdivision vertex* of f, and
- $v_{\{(i,j+1),(i,j)\}}$ is called the *left subdivision vertex* of f.

For $d \in \mathbb{N}$, an *f_0,f_d-face-sequence* in H_ϕ is an alternating sequence of distinct faces and subdivision vertices $(f_0, v_{e_1}, f_1, v_{e_2}, \ldots, v_{e_{d-1}}, f_{d-1}, v_{e_d}, f_d)$ of H_ϕ such that the two edges adjacent to v_{e_h} are on the boundaries of f_{h-1} and f_h for all $h \in [d]$, or equivalently, in $\tilde{G}_k$, the edge e_h bounds the two faces corresponding to f_{h-1} and f_h for all $h \in [d]$. The faces f_h with $h \in [d-1]$ are called the *internal faces* of $(f_0, v_{e_1}, f_1, \ldots, v_{e_d}, f_d)$.

Attachments, Pushing, and Pulling

To construct the graph G, two types of graphs called *attachments* will be added to certain subdivision vertices of H_ϕ. Let v_e be a subdivision vertex of H_ϕ. A *light attachment on* v_e is one new vertex w that is joined to v_e by an edge. A *heavy attachment on* v_e consists of two new, adjacent vertices w_1 and w_2, where w_1 is joined to v_e by an edge. Consider an arbitrary graph $\hat{G}$ that is obtained from H_ϕ by adding heavy and light attachments to subdivision vertices. Note that, as $k \geq 3$ and each cycle in $\hat{G}$ is a cycle in H_ϕ, in any embedding of $\hat{G}$, each face $\hat{f}$ corresponds to exactly one face f of the embedding of H_ϕ such that each vertex on the boundary of f is also on the boundary of $\hat{f}$. Hence, it is not necessary to distinguish between faces of $\hat{G}$ and H_ϕ. In the following, unless specified differently, the term face always refers to a small face of H_ϕ or some graph $\hat{G}$ that is obtained from H_ϕ by adding light and heavy attachments. Define $\ell := 10$. An embedding of a graph $\hat{G}$ with $H_\phi \subseteq \hat{G}$ is said to be *feasible* if each small face of $\hat{G}$ contains at most ℓ vertices of $\hat{G}$, and no vertex in $V(\hat{G}) \setminus V(H_\phi)$ is embedded in the large face of H_ϕ. The aim of the proof is to construct a graph G_ϕ that has a feasible embedding if and only if ϕ is satisfiable. Observing that the boundary of each small face of H_ϕ contains 8 vertices immediately implies the following claim.

Claim 3.38.
Let $\hat{G}$ be an arbitrary graph obtained by adding light and heavy attachments to H_ϕ and consider a feasible embedding of $\hat{G}$. Then, in each small face,

- *there are at most two light attachments and no heavy attachments embedded, or*
- *there are no light attachments and at most one heavy attachment embedded.*

In particular, no small face has a heavy and a light attachment embedded in it.

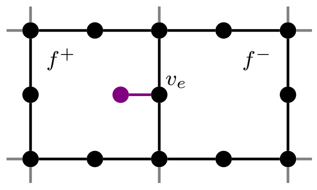
Consider a graph $\hat{G}$ obtained from H_ϕ by adding heavy and light attachments to subdivision vertices and let $F := (f_0, v_{e_1}, f_1, \dots, v_{e_d}, f_d)$ be a face-sequence for some integer $d \geq 1$ such that, in $\hat{G}$, there is exactly one attachment on each subdivision vertex v_{e_h} for all $h \in [d]$. An embedding of $\hat{G}$ is said to *push* along F if the attachment on v_{e_h} is embedded in f_h for all $h \in [d]$, and it is said to *pull* along F if the attachment on v_{e_h} is embedded in f_{h-1} for all $h \in [d]$. Denote by $\hat{G}_0$ the graph obtained from $\hat{G}$ by removing all attachments, except the ones on the subdivision vertices v_{e_h} for all $h \in [d]$. The face-sequence F is said to be *forced to push* if every feasible embedding of $\hat{G}_0$, where the attachment on v_{e_1} is embedded in f_1, is pushing along F. The next claim follows immediately from Claim 3.38.

Claim 3.39.
For $d \geq 1$, consider a face-sequence $F = (f_0, v_{e_1}, f_1, \dots, v_{e_d}, f_d)$ in H_ϕ. Let $\hat{G}$ be an arbitrary graph obtained by adding one attachment to each subdivision vertex v_h with $h \in [d]$. Assume that if, for $h \in [d-1]$, there is a light attachment added to v_h, then there is a heavy attachment added to v_{h+1}. Then, F is forced to push.

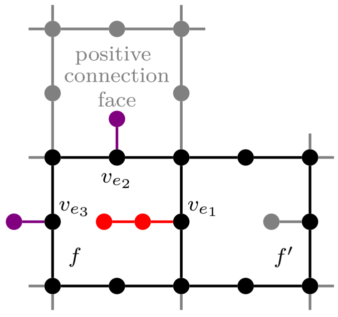
Next, some gadgets are defined, which will then be added to H_ϕ in order to obtain the graph G_ϕ. Each gadget will require a certain number of faces in whose boundary some attachments are added. When the boundary of a face has been modified according to a gadget Y, the face is called a Y *face*. In the following figures, light attachments are colored violet and heavy attachments are colored red. Edges and vertices that do not belong to the displayed gadget are colored gray.

Variable Gadgets

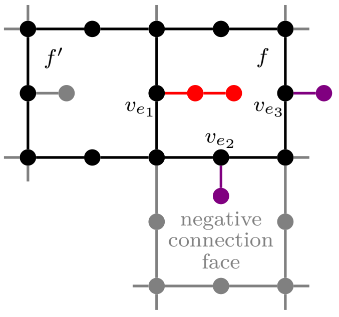
A *variable gadget* for a Boolean variable x consists of a decision gadget and several bifurcation gadgets, which are described now. The *decision gadget* consists of two small faces that are next to each other with a light attachment added to the subdivision vertex that they have in common, see Figure 3.19a). More precisely, let v_e be a subdivision vertex such that e is a vertical edge of $\tilde{G}_k$ that is not on the boundary of the large face of $\tilde{G}_k$. Let f^+ be the face whose right subdivision vertex is v_e and let f^- be the face whose left subdivision vertex is v_e. The faces f^+ and f^- are called the *positive face* and the *negative face* of the variable x, respectively. The decision gadget is to add a light attachment to v_e. This light attachment that is added to v_e is called the *decision attachment of* x. A *positive bifurcation gadget* consists of two faces with one heavy and two light attachments, see Figure 3.19b). More precisely, consider a subdivision vertex v_{e_1} of an edge e_1 such that e_1 is a vertical edge of $\tilde{G}_k$ that is not on the boundary of the large face of $\tilde{G}_k$. The positive bifurcation gadget consists of the faces f and f' whose right subdivision vertex is v_{e_1} and whose left subdivision vertex is v_{e_1}, respectively. A heavy attachment is added to v_{e_1}, one light attachment is added to the top subdivision vertex v_{e_2} of f, and another light attachment is added to the left subdivision vertex v_{e_3} of f. The positive bifurcation gadget will be used in a way that there is a light attachment added to the right subdivision vertex of f' and this light attachment belongs to another gadget, which can be either another positive bifurcation gadget or a decision gadget. The face whose bottom subdivision vertex is v_{e_2} is called the *positive connection face* of the positive bifurcation gadget.



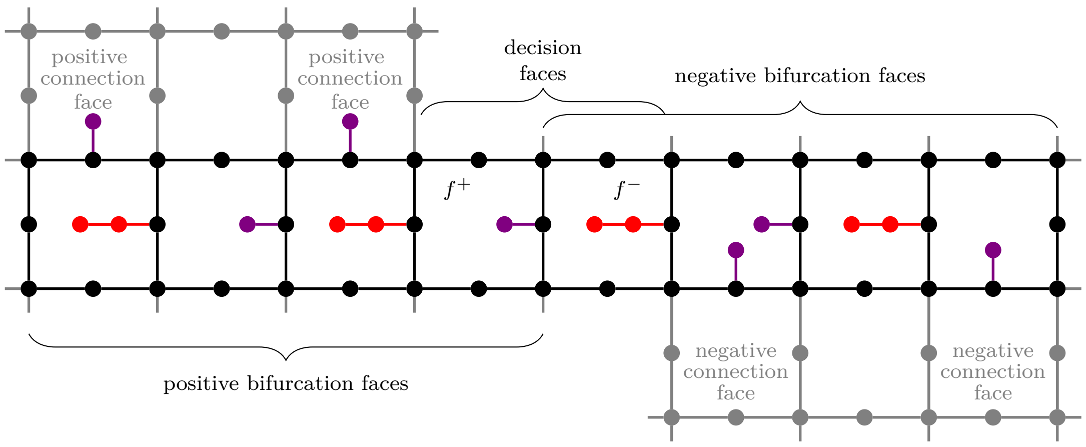
a) A decision gadget.



b) A positive bifurcation gadget.



c) A negative bifurcation gadget.



d) Complete variable gadget, negative embedding.

Figure 3.19: The variable gadget.

Similarly, there is a *negative bifurcation gadget*, which is obtained by rotating the positive bifurcation gadget around 180 degrees, see Figure 3.19c). Also, the negative bifurcation gadget defines a *negative connection face*. A face that is either a positive or a negative connection face is called a *connection face*.

Consider a variable $x \in \mathcal{U}$. The variable gadget for x consists of $2\deg^+(x)$ positive bifurcation faces and $2\deg^-(x)$ negative bifurcation faces, where one positive and one negative bifurcation face are also decision faces, arranged as in Figure 3.19d). More precisely, let $(f_1, v_{e_2}, f_2, v_{e_3}, \ldots, v_{e_d}, f_d)$ be a face-sequence with $d := 2\deg^+(x) + 2\deg^-(x)$ such that v_{e_h} is the right subdivision vertex of f_{h-1} and the left subdivision vertex of f_h for all $h \in [d] \setminus \{1\}$. For each $h \in [\deg^+(x)]$, the faces f_{2h-1} and f_{2h} are modified according to the positive bifurcation gadget. For each integer h with $\deg^+(x)+1 \leq h \leq \deg^+(x)+\deg^-(x)$, the faces f_{2h-1} and f_{2h} are modified according to the negative bifurcation gadget, and the faces $f_{2\deg^+(x)}$ and $f_{2\deg^+(x)+1}$ are additionally modified according to the decision gadget. Furthermore, the light attachment on the left subdivision vertex of f_1 as well as the light attachment on the right subdivision vertex of f_d are removed.

Before describing two further gadgets, some properties of the variable gadget are stated in the next claim. The claim uses the following definitions. Consider the graph $\hat{G}$ obtained from H_ϕ by adding one variable gadget corresponding to a Boolean variable x, and denote by f^+ and f^- the positive and the negative face of its decision gadget. An embedding of $\hat{G}$ is *negative at the variable gadget corresponding to x* if it has the following properties:

- the decision attachment of x is embedded in f^+,
- for each positive connection face f, the embedding is pushing along the f^+,f-face-sequence, whose internal faces all are positive bifurcation faces, and
- for each negative connection face f, the embedding is pulling along the f^-,f-face-sequence, whose internal faces all are negative bifurcation faces,

see Figure 3.19d) for an example. Similarly, an embedding of $\hat{G}$ is *positive at the variable gadget corresponding to x* if it has the following properties:

- the decision attachment of x is embedded in f^-,
- for each positive connection face f, the embedding is pulling along the f^+,f-face-sequence, whose internal faces all are positive bifurcation faces, and
- for each negative connection face f, the embedding is pushing along the f^-,f-face-sequence, whose internal faces all are negative bifurcation faces.

It is easy to check that $\hat{G}$ allows an embedding that is negative at the variable gadget corresponding to x as well as an embedding that is positive at the variable gadget corresponding to x. Now, Claim 3.38 and Claim 3.39 imply the following.

Claim 3.40.
Let $\hat{G}$ be the graph obtained from H_ϕ by adding one variable gadget corresponding to a Boolean variable x, and denote by f^+ and f^- the positive and the negative face of its decision gadget.

a) For each positive connection face f of the considered variable gadget, the f^-,f-face-sequence, whose internal faces all belong to the variable gadget, is forced to push.

b) For each negative connection face f of the considered variable gadget, the f^+,f-face-sequence, whose internal faces all belong to the variable gadget, is forced to push.

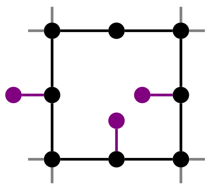
c) The embedding of $\hat{G}$ that is negative at the variable gadget corresponding to x is feasible and, in this embedding, none of the attachments of the variable gadget is embedded in a negative connection face.

d) The embedding of $\hat{G}$ that is positive at the variable gadget corresponding to x is feasible and, in this embedding, none of the attachments of the variable gadget is embedded in a positive connection face.

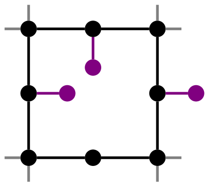
Clause and Wire Gadgets

A *positive clause gadget* consists of one small face f to whose boundary three light attachments are added, as shown in Figure 3.20a). More precisely, a light attachment is added to the following vertices: the left subdivision vertex of f, the bottom subdivision vertex of f, and the right subdivision vertex of f. Similarly, a *negative clause gadget* consists of a small face f to whose boundary three light attachments are added, but instead of using the bottom subdivision vertex of f, the top subdivision vertex of f is used. See also Figure 3.20b).

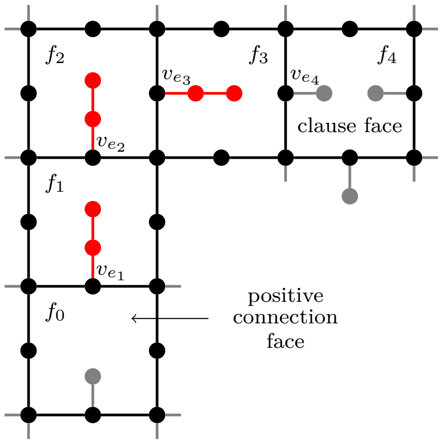
A *wire gadget along a face-sequence F* consists of at least three faces such that the last face of F is a clause face and heavy attachments are added along F, see Figure 3.20c). More precisely, let $d \geq 2$ and consider a face-sequence $F = (f_0, v_{e_1}, f_1, \ldots, v_{e_d}, f_d)$, where f_d is a clause face and such that there is a light attachment on v_{e_d} due to the clause gadget. Then, the wire gadget is to add a heavy attachment to v_{e_h} for all $h \in [d-1]$. When a wire gadget along an f,f'-face-sequence is used, then the face f' has to be a clause face and the face f will be chosen as a connection face, such that the wire gadget joins a variable gadget to a clause gadget. The following claim is a direct consequence of Claim 3.39.



a) A positive clause gadget.



b) A negative clause gadget.



c) A wire gadget joining a positive connection face to a positive clause face.

Figure 3.20: The clause and the wire gadget.

Claim 3.41.
Let $\hat{G}$ be the graph obtained from H_ϕ by adding one clause gadget to a face f_d and one wire gadget along some face-sequence $F = (f_0, v_{e_1}, f_1, \dots, v_{e_d}, f_d)$ with $d \geq 2$.

a) The face-sequence F is forced to push.

b) The embedding that is pulling along F is feasible.

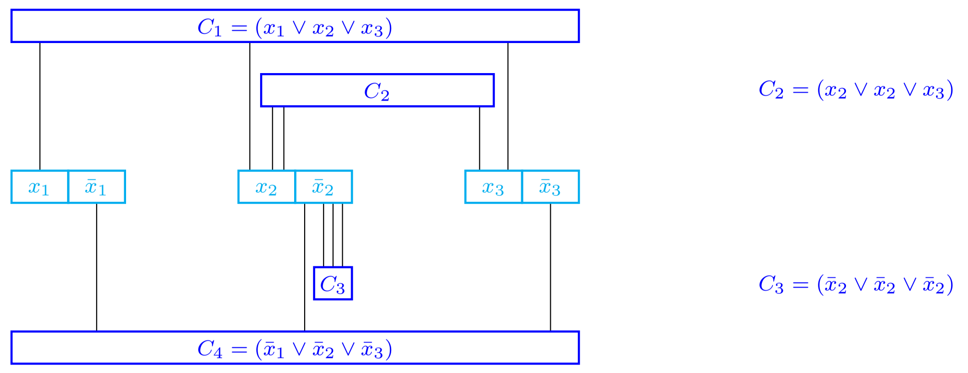
Assembling the Graph G_ϕ

Next, some gadgets will be added to H_ϕ in order to obtain the desired graph G_ϕ. Recall that $k := 6m + n$ and that H_ϕ is a subdivision of a $k \times k$ grid. The variable gadgets will be added one after another to the middle row of H_ϕ. More precisely, let $j_v = \left\lfloor \frac{1}{2}k \right\rfloor$. For each variable $x \in \mathcal{U}$, a variable gadget will be added to small faces whose boundaries have two vertices in $R_{j_v} := \{(i, j_v) \colon i \in [k]\}$ and two vertices in $R_{j_v+1} := \{(i, j_v + 1) \colon i \in [k]\}$. Let f_1 be the small face whose boundary contains the vertices $(1, j_v)$ and $(1, j_v + 1)$ and let f_{k-1} be the small face whose boundary contains the vertices (k, j_v) and $(k, j_v + 1)$. To place the variable gadgets, the $k - 1$ small faces are considered from left to right, i. e., f_1 is considered first and f_{k-1} is considered last. Let $\mathcal{U} = \{x_1, \dots, x_n\}$ and assume that $x_1, \dots, x_n$ is the order in which the variables appear in $\mathcal{R}$. Starting from f_1, a variable gadget for x_1 is added to H_ϕ, then there is one face that does not belong to a gadget, and then a variable gadget for x_2 is added and so on. In total $2 \sum_{x \in \mathcal{U}} \big(\deg^+(x) + \deg^-(x)\big)$ faces are needed for all variable gadgets together and $(n - 1)$ faces are needed in between them. As $\sum_{x \in \mathcal{U}} \big(\deg^+(x) + \deg^-(x)\big)$ counts each vertical line of $\mathcal{R}$ once and each clause in $\mathcal{R}$ touches exactly three vertical lines, the following holds

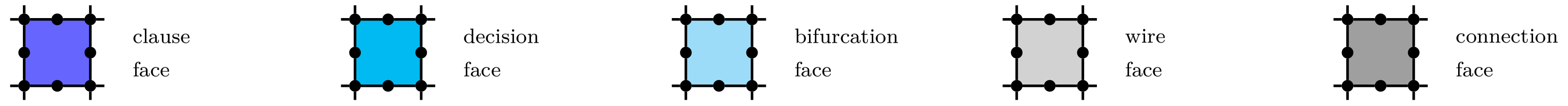
$$2 \sum_{x \in \mathcal{U}} \big(\deg^+(x) + \deg^-(x)\big) + (n - 1) \;=\; 6|\mathcal{C}| + n - 1 \;=\; 6m + n - 1 \;=\; k - 1.$$

Hence, there are enough faces to place all variable gadgets next to each other. Let G_1 be the graph obtained from H_ϕ by adding the variable gadgets in this way.

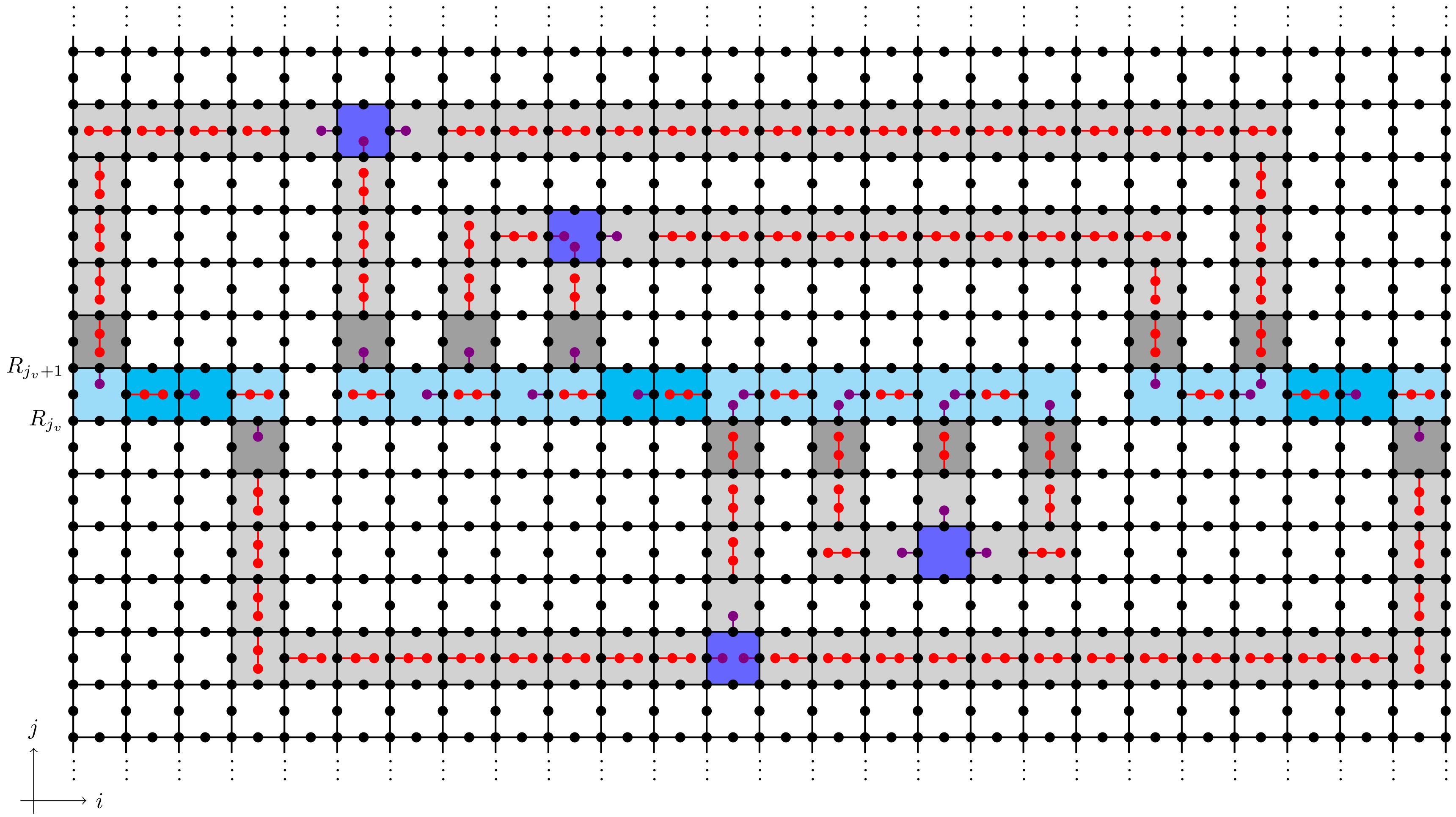
For $d, d' \geq 2$, two face-sequences $(f_0, v_{e_1}, f_1, \dots, v_{e_d}, f_d)$ and $(f'_0, v_{e'_1}, f'_1, \dots, v_{e'_{d'}}, f'_{d'})$ are called *disjoint* if $f_h \neq f'_{h'}$ for all $h \in [d] \cup \{0\}$ and all $h' \in [d'] \cup \{0\}$ and $(f_0, v_{e_1}, f_1, \dots, v_{e_d}, f_d)$ and $(f'_0, v_{e'_1}, f'_1, \dots, v_{e'_{d'}}, f'_{d'})$ are called *disjoint except for the last face* if the only violation to being disjoint is that $f_d = f'_{d'}$. In the


$C_1 = (x_1 \vee x_2 \vee x_3)$
C_2
x_1
$\bar{x}_1$
x_2
$\bar{x}_2$
x_3
$\bar{x}_3$
C_3
$C_4 = (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$
$C_2 = (x_2 \vee x_2 \vee x_3)$
$C_3 = (\bar{x}_2 \vee \bar{x}_2 \vee \bar{x}_2)$

a) Monotone rectilinear representation of ϕ.



b) Colors used in Part c). Observe that every decision face is also a bifurcation face and that every connection face is also a wire face.



c) The graph G_ϕ. Recall that the vertex (i,j) for $i,j \in [k]$ is embedded at the coordinates (i,j), where the horizontal axis refers to the first coordinate and the vertical axis refers to the second coordinate. The embedding corresponds to the satisfying assignment $x_1 = \text{True}$, $x_2 = \text{False}$, and $x_3 = \text{True}$.

Figure 3.21: Example of the graph G_ϕ for $\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_2 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$.

following, f_C is used to denote the face where the clause gadget corresponding to the clause $C \in \mathcal{C}$ is added and for a vertical line L in $\mathcal{R}$ that joins a variable $x \in \mathcal{U}$ to a clause $C \in \mathcal{C}$, we use $\hat{F}_L$ to denote a face-sequence starting in a connection face of the variable gadget corresponding to x and ending in f_C. Using $\mathcal{R}$, one can choose faces f_C in G_1 for all $C \in \mathcal{C}$ and face-sequences $\hat{F}_L$ in G_1 for all vertical lines L in $\mathcal{R}$ such that

- for all distinct $C, C' \in \mathcal{C}$, the faces f_C and $f_{C'}$ are distinct,
- for all distinct lines L, L' in $\mathcal{R}$, the face-sequences $\hat{F}_L$ and $\hat{F}_{L'}$ are disjoint, except when L and L' touch the same clause in $\mathcal{R}$, in which case $\hat{F}_L$ and $\hat{F}_{L'}$ are disjoint except for the last face,
- for each line L in $\mathcal{R}$, the face-sequence $\hat{F}_L$ consists of at least three faces and none of the faces in $\hat{F}_L$ is a variable face,
- for each line L in $\mathcal{R}$ that joins a variable $x \in \mathcal{U}$ to a positive clause $C \in \mathcal{C}$, the face-sequence $\hat{F}_L = (f_0, v_{e_1}, f_1, \dots, v_{e_d}, f_d)$ starts in a face f_0 that is a positive connection face of the variable gadget corresponding to x, ends in the clause face f_C, i. e., $f_d = f_C$, and v_{e_d} is not the top subdivision vertex of f_d,
- for each line L in $\mathcal{R}$ that joins a variable $x \in \mathcal{U}$ to a negative clause $C \in \mathcal{C}$, the face-sequence $\hat{F}_L = (f_0, v_{e_1}, f_1, \dots, v_{e_d}, f_d)$ starts in a face f_0 that is a negative connection face of the variable gadget corresponding to x, ends in the clause face f_C, i. e., $f_d = f_C$, and v_{e_d} is not the bottom subdivision vertex of f_d, as well as
- for all $x \in \mathcal{U}$, for each connection face f of the variable gadget corresponding to x, there is a unique line L in $\mathcal{R}$ such that the face sequence $\hat{F}_L = (f_0, v_{e_1}, \dots, f_d)$ starts in f, i. e., $f_0 = f$.

Indeed, $j_v = \left\lfloor \frac{1}{2} k \right\rfloor \geq 3m$ and hence, there is enough space for each clause $C \in \mathcal{C}$. Moreover, the total number of connection faces equals the number of vertical lines in $\mathcal{R}$ and, hence, all required properties can be satisfied. Let G_2 be the graph obtained from G_1 by placing a positive clause gadget in the face f_C for each positive clause $C \in \mathcal{C}$ as well as a negative clause gadget in the face f_C for each negative clause $C \in \mathcal{C}$. Furthermore, let G_ϕ be the graph obtained from G_2 by adding a wire gadget along the face-sequence $\hat{F}_L$ for each vertical line L in $\mathcal{R}$, see Figure 3.21 for an example. Observe that, for each subdivision vertex v_e of H_ϕ, at most one attachment is added to v_e. Hence, G_ϕ has size polynomial in $k = 6m + n$ and also polynomial in $n + m$, which is a lower bound on the size of ϕ.

For each vertical line L in $\mathcal{R}$ that joins a variable $x \in \mathcal{U}$ to a positive clause $C \in \mathcal{C}$, let F_L be the face-sequence $(f_0, v_{e_1}, f_1, \dots, v_{e_d}, f_d)$ where $f_d = f_C$ and f_0 is the negative face of the decision gadget corresponding to x, which is obtained from $\hat{F}_L$ by adding f_0 and positive bifurcation faces of the variable gadget corresponding to x. Similarly, for each vertical line L in $\mathcal{R}$ that joins a variable $x \in \mathcal{U}$ to a negative clause $C \in \mathcal{C}$, let $F_L = (f_0, v_{e_1}, f_1, \dots, v_{e_d}, f_d)$ be the face-sequence where $f_d = f_C$ and f_0 is the positive face of the decision gadget corresponding to x, which is obtained from $\hat{F}_L$ by adding f_0 and negative bifurcation faces of the variable gadget corresponding to x. It is easy to verify that, for each attachment in G_ϕ, there is a line L in $\mathcal{R}$ such that the attachment is added to a subdivision vertex in the face-sequence F_L. The following claim follows from Claim 3.40a), Claim 3.40b), and Claim 3.41a).

Claim 3.42.
For each vertical line L in $\mathcal{R}$, the face-sequence F_L in G_ϕ is forced to push.

The “if and only if” statement

To finish the proof, it remains to show that ϕ is satisfiable if and only if there is a feasible embedding of G_ϕ. Assume that ϕ is satisfiable and denote by $T : \mathcal{U} \to \{\textsc{False}, \textsc{True}\}$ a satisfying assignment of ϕ. Consider an embedding of G_ϕ that is obtained from the fixed embedding of H_ϕ by embedding the attachments according to the following rules:

(i) For each variable $x \in \mathcal{U}$ with $T(x) = \text{True}$, the attachments of the variable gadget corresponding to x are embedded as in the positive embedding of the variable gadget corresponding to x.
(ii) For each variable $x \in \mathcal{U}$ with $T(x) = \text{False}$, the attachments of the variable gadget corresponding to x are embedded as in the negative embedding of the variable gadget corresponding to x.
(iii) For each line L in $\mathcal{R}$, that joins a variable $x \in \mathcal{U}$ to some clause in $\mathcal{C}$, the embedding is pulling along $\hat{F}_L = (f_0, v_{e_1}, f_1, \dots, v_{e_d}, f_d)$ if none of the attachments of the variable gadget corresponding to x is embedded in the connection face f_0 due to (i) or (ii) and otherwise the embedding is pushing along $\hat{F}_L$.

Clearly, given a satisfying assignment such an embedding always exists. For an example, see Figure 3.21c). Next, it is argued that every embedding of G_ϕ that satisfies (i)-(iii) is feasible. To do so, fix such an embedding of G_ϕ and call a face f of G_ϕ *feasible* if it contains at most $\ell = 10$ vertices. Due to Claim 3.40c) and Claim 3.40d), all variable faces are feasible. Furthermore, (iii) ensures that all connection faces are feasible and due to Claim 3.41 all wire faces that are neither connection faces nor clause faces are feasible. Consider a positive clause $C \in \mathcal{C}$. As the assignment T is a satisfying assignment for ϕ, there is a variable $x \in \mathcal{U}$ with $T(x) = \text{True}$ that is used in C. So there is a vertical line L that joins x to C in $\mathcal{R}$. Let $\hat{F}_L = (f_0, v_{e_1}, f_1, \dots, v_{e_d}, f_d)$ and note that f_0 is a positive connection face of the variable gadget corresponding to x. Then, as the variable gadget corresponding to x is embedded according to (i), there is no attachment of the variable gadget corresponding to x embedded in f_0 by Claim 3.40d) and the embedding of G_ϕ is pulling along $\hat{F}_L$ according to (iii). Therefore, the attachment on v_{e_d} is embedded in f_{d-1} and the clause face $f_C = f_d$ is feasible. Similarly, one can show that each face f_C corresponding to a negative clause $C \in \mathcal{C}$ is feasible. Consequently, the embedding of G_ϕ is feasible.

Now, consider a feasible embedding of G_ϕ that is an extension of the fixed embedding of H_ϕ. To define a truth assignment $T : \mathcal{U} \to \{\text{False}, \text{True}\}$ set $T(x) = \text{True}$ if and only if the decision attachment corresponding to x is embedded in the negative face of the decision gadget corresponding to x. For a contradiction, assume that there is a clause $C \in \mathcal{C}$ that is not satisfied by the assignment T, i. e., C evaluates to False when plugging in the values for the variables according to T. Assume that C is a positive clause. The following is easy to adjust for a negative clause. Let x be an arbitrary variable used in C and denote by L_x a line in $\mathcal{R}$ that joins x to C. Then $T(x) = \text{False}$ and, hence, the decision attachment corresponding to x is embedded in the positive face of the decision gadget corresponding to x. Let $F_{L_x} = (f_0, v_{e_1}, f_1, \dots, v_{e_d}, f_d)$. Thus, $f_d = f_C$ and f_0 and f_1 are the negative and the positive face of the decision gadget corresponding to x, respectively. Furthermore, the decision attachment of x has been added to v_{e_1} and is embedded in f_1. By Claim 3.42, the attachment of v_{e_d} must be embedded in $f_d = f_C$. As x was an arbitrary variable used in C and C contains exactly three literals by assumption, three attachments are embedded in f_C. Then f_C is not feasible, which contradicts that the embedding of G_ϕ is feasible. Consequently, C must be satisfied and T is a satisfying assignment for ϕ.

Strengthening the Proof and a Remark on UPM Scheduling

After completing the proof of Theorem 3.37, it is discussed here, how to strengthen the proof to obtain Theorem 3.26, i. e., it is shown that the HEG Problem is hard for $\ell = 6$. The graph H_ϕ, the parameter ℓ, and the attachments can be defined differently as long as they still satisfy the properties in Claim 3.38 and Claim 3.39. It is easy to check that the attachments defined in Figure 3.22, $\ell = 6$, and the graph H_ϕ that is a $k \times k$ grid where k is defined as above in the proof of Theorem 3.37, have these properties. Therefore, the HEG Problem is NP-hard for $\ell = 6$ and Theorem 3.26 follows.

Observe that the proof of Theorem 3.37 also shows that the following restricted version of the UPM Scheduling Problem is NP-hard. Consider m' machines and n' jobs of length 1 and 2, where each

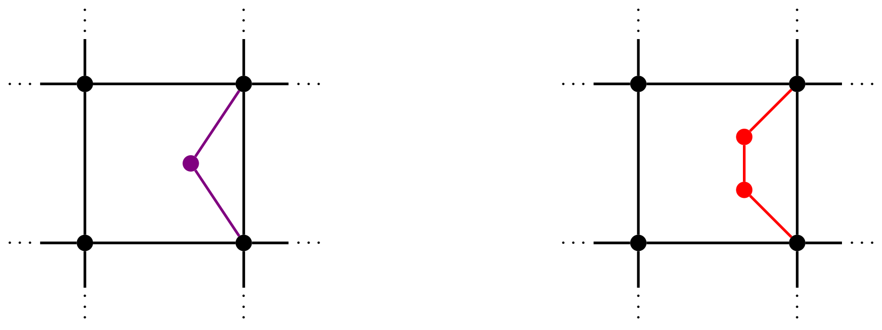


Figure 3.22: Modifying the attachments in the proof of Theorem 3.37. Each figure shows one small face of H_ϕ. The subgraph H_ϕ is colored black, the vertices and edges of the light and heavy attachments are colored purple and red, respectively.

job can be processed by at most two of the machines. The Restricted Scheduling Problem asks whether there is a schedule of makespan ℓ', where $\ell' \in \mathbb{N}$ is part of the input.

Modifying the Proof to Show Theorem 3.28

The proof of Theorem 3.37 can be modified to show that it is NP-hard to approximate the Min HEG Problem within α for every $\alpha < \frac{3}{2}$. Recall that, in Section 3.3.1, we argued that it is NP-hard to approximate the Min HEG Problem within α for every $\alpha < \frac{7}{6}$. This fraction results from the answers ℓ for the Min HEG Problem corresponding to no- and yes-instances, which are at least 7 and at most 6, respectively. Here, the ratio of these numbers is increased by modifying the reduction used to prove Theorem 3.37. Consider an instance of the Planar Monotone 3-SAT Problem and denote by ϕ the underlying 3-SAT formula. Let k be the integer and let G_ϕ and H_ϕ be the graphs as defined above in the proof of Theorem 3.37 but with the following modified attachments. Fix an integer $s \geq 2$. The light attachment is now defined to be a path on s vertices, one of whose leaves is joined to a subdivision vertex of H_ϕ and the heavy attachment is now defined to be a path on $2s$ vertices, one of whose leaves is joined to a subdivision vertex of H_ϕ, see Figure 3.23. It is easy to see that, if ϕ is satisfiable, then G_ϕ allows an embedding where each small face of H_ϕ contains at most $2s + 8$ vertices. So when applying an α-approximation to the instance (k, G_ϕ, H_ϕ), a number $\ell_{\text{app}} \leq \alpha \cdot (2s + 8)$ is computed. Furthermore, when ϕ is not satisfiable, then any embedding of G_ϕ contains a small face f

- that contains at least three light attachments,
- that contains at least one heavy and one light attachment, or
- that contains at least two heavy attachments.

So this face f contains at least $3s + 8$ vertices of G and, when given the instance (k, G_ϕ, H_ϕ) as input, an α-approximation would yield a number $\ell_{\text{app}} \geq 3s + 8$. Hence, an α-approximation for the Min HEG Problem can be used to solve the Planar Monotone 3-SAT Problem when $\alpha \cdot (2s + 8) < 3s + 8$, or equivalently when

$$\alpha < \frac{3s+8}{2s+8} \xrightarrow{s \to \infty} \frac{3}{2}.$$

Recalling that the Planar Monotone 3-SAT Problem is NP-complete as stated in Theorem 3.36 now implies the desired result, i. e., Theorem 3.28.

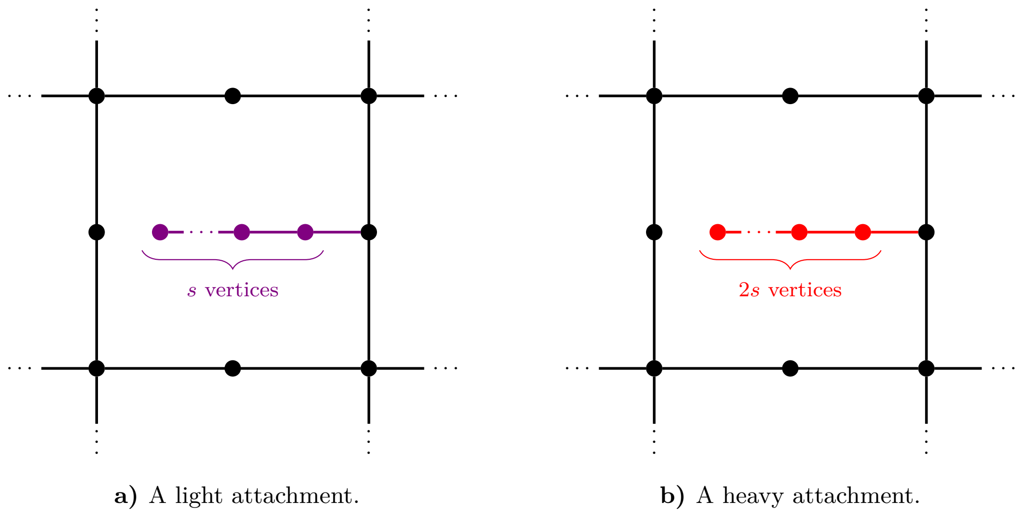


Figure 3.23: Modifying the attachments in the proof of Theorem 3.37. Each figure shows one small face of H_ϕ. The subgraph H_ϕ is colored black, the vertices and edges of the light and heavy attachments are colored purple and red, respectively.

3.3.4 Proof of a Hardness Result for the HGM Problem

Here, the proof of Theorem 3.31, which states that the HGM Problem is NP-complete even when restricted to instances (k, ℓ, G) where G is uniquely embeddable, is presented. As in Section 3.3.3, the reduction is from Planar Monotone 3-SAT. For a planar, monotone 3-SAT formula ϕ, a graph G_ϕ and two integers k, ℓ will be defined. It will be clear that G_ϕ contains a $k \times k$ grid as a minor, but it will depend on the satisfiability of ϕ whether the additional properties required, such that (k, ℓ, G_ϕ) is a yes-instance of the HGM Problem, can be met. The idea is to start with a $k \times k$ grid and, for each variable, to chose a column and delete an edge joining two vertices in that column. Some new edges and new vertices are added, such that each column of the initial grid, from which an edge was deleted, has two choices for making a detour, which will correspond to the variable being True and False. Such a detour will then force other columns and rows to make detours in order to satisfy that each small face contains at most ℓ vertices. Each detour pulls on the boundary of a face, which makes the face larger and forces it to contain more vertices. So, in the following reduction, pulling can be interpreted as the False-information being spread through the graph. Note that this is the opposite of the proof of Theorem 3.37, where pushing can be interpreted as the False-information being spread through the given grid. Again, there will only be modifications in certain parts of the grid and, if the resulting graph contains a graph H that is a minimal graph containing a $k \times k$ grid as a minor, then H must look almost as the grid with which the construction started.

The following lemma will be used. Consider a graph $G = (V, E)$ and let $Z_1, Z_2 \subseteq V$. A set $S \subseteq V$ *separates* Z_1 *from* Z_2 in G if $G - S$ contains no Z_1,Z_2-path. The smallest integer h such that there is a set $S \subseteq V$ of size h that separates Z_1 from Z_2 in G is called the *minimum number of vertices separating* Z_1 *from* Z_2 in G.

Lemma 3.43 (Menger's Theorem, see Theorem 3.3.1 in [Die12]).
Let $G = (V, E)$ be a graph. For every $Z_1, Z_2 \subseteq V$, the minimum number of vertices separating Z_1 from Z_2 in G is equal to the maximum number of disjoint Z_1,Z_2-paths in G.

The remainder of the section will be concerned with the proof of Theorem 3.31. First, some notation is defined, then gadgets are introduced and some properties of the gadgets are proved, and finally the if-and-only-if-statement of the reduction is shown.

Notation

Consider an instance of the Planar Monotone 3-SAT Problem, i. e., a monotone rectilinear representation $\mathcal{R}$ of a 3-SAT formula $\phi = (\mathcal{U}, \mathcal{C})$. Define $n := |\mathcal{U}|$ and $m := |\mathcal{C}|$. Recall that the rectangles in $\mathcal{R}$ are identified with the corresponding variables and clauses in $\mathcal{C} \cup \mathcal{U}$. As in the proof of Theorem 3.37, the assumptions discussed in the end of Section 3.3.2 are used, i. e., each variable in $\mathcal{U}$ is used at least once as a positive literal and at least once as a negative literal in ϕ, each clause in $\mathcal{C}$ is a disjunction of three not necessarily distinct literals, and, in $\mathcal{R}$, each variable x is split into a positive and a negative half.

Set $k := 6m + 2n + 3$ and $\ell := 8$. Denote by $\tilde{G}_k$ the $k \times k$ grid and recall that the vertex set of $\tilde{G}_k$ is $\{(i,j)\colon\ i \in [k], j \in [k]\}$. For $i \in [k]$, let $\tilde{C}_i := \{(i,j)\colon\ j \in [k]\}$ be the i^{th} *column of* $\tilde{G}_k$, and, for $j \in [k]$, let $\tilde{R}_j := \{(i,j)\colon\ i \in [k]\}$ be the j^{th} *row of* $\tilde{G}_k$. Furthermore, let G_0 be the graph obtained from the $k \times k$ grid $\tilde{G}_k$ by subdividing each edge once, except for the edges on the boundary of the large face of $\tilde{G}_k$. Clearly, G_0 is a minimal graph containing a $k \times k$ grid as a minor and, hence, G_0 is uniquely embeddable due to Remark 3.15. In the following, the embedding of G_0, where the vertex (i,j) for $i \in [k]$ and $j \in [k]$ is embedded at the point (i,j) in a coordinate system whose horizontal axis refers to the first coordinate and whose vertical axis refers to the second coordinate is considered and from now on G_0 refers to the plane graph with this embedding. Similarly to Section 3.3.3, a vertex of G_0 that subdivides an edge e of $\tilde{G}_k$ is called a *subdivision vertex* and is denoted by v_e. All other vertices of G_0 are called *grid vertices* as they are also in $V(\tilde{G}_k)$. Furthermore, the terms bottom, right, top, and left subdivision vertex of a small face of G_0 are used as in Section 3.3.3. Again, face-sequences of the graph G_0 are used, which are alternating sequences of small faces of G_0 and subdivision vertices of G_0 such that each subdivision vertex is on the boundary of the face preceding it and the face succeeding it. When modifying the graph G_0 and its embedding, some faces will be split into several faces, but, if a face-sequence is considered in this proof, it refers to the faces of G_0. Denote by B_{large} the cycle in G_0 that bounds the large face of G_0, i. e.,

$$B_{\text{large}} = ((1,1),(2,1),\ldots,(k,1),(k,2),\ldots,(k,k),(k-1,k),\ldots,(1,k),(1,k-1),\ldots,(1,2)).$$

The *boundary distance* of a small face f of G_0 is the length of a shortest u,w-path such that u is on the boundary of f and w belongs to B_{large}. For example, the small face of G_0 whose boundary contains the vertices $(2,2)$ and $(3,3)$ has boundary distance 2 and the small face of G_0 whose boundary contains the vertices $(k-1,k-1)$ and (k,k) has boundary distance 0.

Consider a graph H that is a minimal graph containing a $k \times k$ grid as a minor. Next, columns and rows of H are defined. To do so, consider a partition of $V(H)$ into sets $M_{i,j}$ with $i,j \in [k]$ as in Proposition 3.24. For each vertex (i,j) of $\tilde{G}_k$ with $\deg_{\tilde{G}_k}((i,j)) = 3$, there is a unique vertex $v_{i,j}$ in $M_{i,j}$ with $\deg_H(v_{i,j}) = 3$ due to Proposition 3.16. Let $v_{1,1}$, $v_{1,k}$, $v_{k,1}$, and $v_{k,k}$ be arbitrary vertices in $M_{1,1}$, $M_{1,k}$, $M_{k,1}$, and $M_{k,k}$, respectively. For $i \in [k]$, let the i^{th} column C_i^H of H be the vertex set of the unique $v_{i,1}$,$v_{i,k}$-path in H that uses only vertices in $M_{i,j}$ with $j \in [k]$. For $j \in [k]$, let the j^{th} row R_j^H of H be the vertex set of the unique $v_{1,j}$,$v_{k,j}$-path in H that uses only vertices in $M_{i,j}$ with $i \in [k]$. Such paths exist and are unique due to Proposition 3.24. Furthermore, by construction, the sets $C_1^H, \ldots, C_k^H$ are pairwise disjoint and so are the sets $R_1^H, \ldots, R_k^H$. Each column and each row induces a path in H. Such paths are called *grid-paths* in H.

Consider a plane graph $\hat{G}$ with $G_0 \subseteq \hat{G}$ and a subgraph $H \subseteq \hat{G}$. If H is a minimal graph containing a $k \times k$ grid as a minor, then a small face f of H is called *feasible with respect to* $\hat{G}$ if, in the drawing of the plane graph $\hat{G}$, at most ℓ vertices of $V(\hat{G})$ are embedded in f. Recall that each vertex that is on the boundary of f contributes to the number of vertices embedded in f. The graph H is called a *feasible subgraph of* $\hat{G}$ if H is a minimal graph containing a $k \times k$ grid as a minor, every small face of H is feasible with respect to $\hat{G}$, and no vertex from $V(\hat{G}) \setminus V(H)$ is embedded in the large face of H. The subgraph H is called *normal* if

- H is a minimal graph containing a $k \times k$ grid as a minor,
- there is a partition as in Proposition 3.24 such that the resulting columns $C_1^H, \dots, C_k^H$ of H satisfy $\tilde{C}_i \subseteq C_i^H$ for all $i \in [k]$ and such that the resulting rows $R_1^H, \dots, R_k^H$ of H satisfy $\tilde{R}_j \subseteq R_j^H$ for all $j \in [k]$, as well as
- the boundary of the large face of H is the cycle B_{large}.

When considering a normal graph H, then we use $C_1^H, \dots, C_k^H$ and $R_1^H, \dots, R_k^H$ to denote a choice of columns and rows that satisfy these properties. Assume that H is normal. Using the columns and the rows of H, a bijection between the small faces of G_0 and the small faces of H is defined now. Let f be a small face of G_0. Then, there are unique integers $i, j \in [k-1]$ such that the grid vertices (i, j) and $(i+1, j+1)$ are on the boundary of f. Let f' be the unique small face of H with the property that each vertex on the boundary of f' is in $C_i^H \cup C_{i+1}^H \cup R_j^H \cup R_{j+1}^H$. Then, f' is referred to as the *face of H that corresponds to f*.

Basic Modifications, Pulling, and Relaxing

Next, some operations that modify the graph G_0 are described. For a small face f of G_0 with boundary distance at least two, *splitting f vertically* means to insert a new edge in G_0 that is embedded in f and that joins the top subdivision vertex of f to the bottom subdivision vertex of f. Similarly, for a small face f of G_0 with boundary distance at least two, *splitting f horizontally* means to insert a new edge in G_0 that is embedded in f and that joins the left subdivision vertex of f to the right subdivision vertex of f. For a subdivision vertex v_e of G_0, *replacing v_e by its original edge* roughly means to reverse the subdivision that created v_e. More precisely, let $e \in E(\tilde{G}_k)$ be an edge that was subdivided when constructing G_0, then, in the graph G_0, replacing v_e by its original edge means to remove v_e as well as its incident edges e' and e'' from G_0, to add the edge e to the edge set of G_0, and, in the embedding of G_0, to draw e where e' and e'' used to be drawn. Let f be a face of G_0 and let $\hat{G}$ be the graph obtained from G_0 by replacing some subdivision vertices of G_0 by their original edges. Then, $\hat{G}$ contains a unique face $\hat{f}$ such that each grid vertex on the boundary of f is also on the boundary of $\hat{f}$. In the following, the face f and the face $\hat{f}$ are considered to be the same face or, more precisely, we say that f is a face of $\hat{G}$ and that the face f has been modified when constructing $\hat{G}$.

Consider a graph $\hat{G}$ obtained from G_0 by these modifications. Let v_e be a subdivision vertex of G_0 and denote by u and w the neighbors of v_e in G_0. Let $H \subseteq \hat{G}$ be a normal graph and let P be the grid-path of H that contains u and w. Observe that P exists as u and w are adjacent in $\tilde{G}_k$. We say that *P uses either e or v_e* if either v_e is a vertex of $\hat{G}$ and P uses v_e and both its incident edges to join u to w or if v_e has been replaced by its original edge, which is e, when constructing $\hat{G}$ and P uses e to join u to w. Roughly speaking, the next claim says that, in a normal and feasible subgraph of $\hat{G}$, a grid-path P can only make a detour compared to the corresponding grid-path P_0 in G_0 if all faces of G_0 containing an edge from $E(P_0) \setminus E(P)$ on their boundary have been modified.

Claim 3.44.
Consider a subdivision vertex v_e of G_0, let f_1 and f_2 be the two small faces of G_0 whose boundaries contain v_e, and denote by u and w the neighbors of v_e in G_0. Let $\hat{G}$ be a plane graph obtained from G_0 by replacing some subdivision vertices of G_0 by their original edges and by embedding new vertices and edges

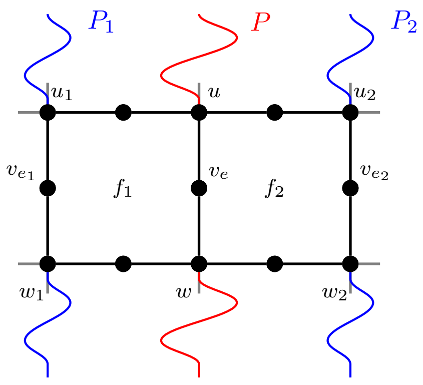


Figure 3.24: Proof of Claim 3.44.

in some small faces of G_0 such that each new edge is incident with at most one grid vertex. Consider a normal subgraph $H \subseteq \hat{G}$ and let P be the grid-path of H that contains u and w.

a) If f_1 and f_2 are also faces of $\hat{G}$ then P uses either v_e or e.

b) If H is a feasible subgraph of $\hat{G}$ and there is an $h \in [2]$ such that f_h has boundary distance at least 2 and f_h has not been modified when constructing $\hat{G}$, then P uses v_e.

c) If one of the following holds

- *P is induced by a column of H and, for all $h \in [2]$, when constructing $\hat{G}$ from G_0, the face f_h has not been modified other than splitting f_h horizontally and replacing some subdivision vertices that are on the boundary of f_h by their original edges,*
- *P is induced by a row of H and, for all $h \in [2]$, when constructing $\hat{G}$ from G_0, the face f_h has not been modified other than splitting f_h vertically and replacing some subdivision vertices that are on the boundary of f_h by their original edges,*

then P uses the subdivision vertex v_e.

Proof. Let v_e, f_1, f_2, u, w, $\hat{G}$, H, and P be as in the statement. As u and w are grid vertices that are adjacent in $\tilde{G}_k$, there is indeed a grid-path P that contains u and w. For $h \in [2]$, denote by u_h and w_h the two grid vertices on the boundary of f_h that are not contained in P and assume without loss of generality that u and u_h are adjacent in $\tilde{G}_k$ as well as that w and w_h are adjacent in $\tilde{G}_k$, see Figure 3.24. Moreover, for $h \in [2]$, let P_h be the grid-path of H that contains u_h and w_h, and let e_h be the edge that joins u_h to w_h in $\tilde{G}_k$.

a) As H is planar and the grid-paths P_1, P, and P_2 are vertex-disjoint, P_1, P, and P_2 cannot cross. If, for all $h \in [2]$, the grid-path P_h uses either v_{e_h} or e_h, then P uses either v_e or e. Otherwise, there is an $h \in [2]$ such that P_h uses neither v_{e_h} nor e_h to join u_h to w_h. Let P'_h be the subpath of P_h that joins u_h to w_h. Then, P'_h and either the edge e_h or the edges incident to v_{e_h} form a cycle in $\hat{G}$. Now, all vertices of P are on the same side of this cycle, as P could only cross the cycle in the vertex v_{e_h} but, as none of the neighbors of v_{e_h} can be in P, there is no edge incident to v_{e_h} that is embedded in f_h and can be used by P. Consequently, P uses either v_e or e.

b) Assume there is an $h \in [2]$ such that f_h has boundary distance at least 2 and f_h has not been modified when constructing $\hat{G}$. Then, there is a face f'_h of H such that each vertex on the boundary of f_h is embedded in f'_h. Since no new edge was embedded in f_h and each new edge is incident to at most one grid vertex, there is no edge in $\hat{G}$ that is not in G_0 and that joins two vertices on the boundary of f_h. As, in $\hat{G}$, there are 8 vertices on the boundary of f_h and f'_h is a feasible face of H with respect to $\hat{G}$, the boundary of f'_h must be the same as the boundary of f_h. Hence, P uses v_e.

c) The proof is similar to Part a) and follows by observing that, for $h \in [2]$, the path P might now be able to cross the cycle, that consists of P'_h and the subdivision vertex v_{e_h} and its incident edges, once in v_{e_h}. However, P would need to cross this cycle twice as u and w are on the same side of the cycle. □

Consider a plane graph $\hat{G}$ with $G_0 \subseteq \hat{G}$ and a normal graph $H \subseteq \hat{G}$. For an arbitrary integer $d \geq 1$, let $F := (f_0, v_{e_1}, f_1, \ldots, v_{e_d}, f_d)$ be a face-sequence in G_0. For $h \in [d] \cup \{0\}$, denote by u_h and w_h the two neighbors of v_{e_h} in G_0. If $d \geq 2$ and f_h is a small face of H for all $h \in [d-1]$, then we say that H *relaxes along* F. In the case when $d = 1$, we say that H *relaxes along* F if the grid-path of H that uses u_1 and w_1 uses either v_{e_1} or e_1. Note that, if H relaxes along F, then the grid-path of H that contains u_h and w_h uses either v_{e_h} or e_h for each $h \in [d]$. Moreover, we say that H *pulls along* F if, for all $h \in [d]$, the grid-path P of H that contains u_h and w_h neither contains v_{e_h} nor e_h and, between u_h and w_h, the grid-path P uses only vertices embedded in f_{h-1}. Observe that, as H is normal, there is indeed a column or a row of H that contains both u_h and w_h for every $h \in [d]$, since u_h and w_h are grid vertices that are adjacent in $\tilde{G}_k$. Next, the expression "forces to pull" is defined. Roughly speaking, this means that, if a normal graph H pulls along (f_0, v_{e_1}, f_1), then it has to pull along the entire face-sequence F in order to be feasible except for the face f_d. More precisely, denote by $R(G_0)$ the set of faces of G_0. Let $\hat{G}_0$ be the plane graph obtained from $\hat{G}$ by removing each vertex from $V(\hat{G}) \setminus V(G_0)$ that is embedded in a face f of G_0 with $f \in R(G_0) \setminus \{f_0, f_1, \ldots, f_d\}$ and modifying the embedding of $\hat{G}$ accordingly to obtain an embedding of $\hat{G}_0$. Consider the set $\mathscr{H}$ of normal graphs $H' \subseteq \hat{G}_0$, with the following properties:

- The grid-path of H', that contains u_1 and w_1, contains neither v_{e_1} nor e_1 and uses between u_1 and w_1 only vertices embedded in f_0.
- Every small face of H', except possibly the face of H that corresponds to f_d, is feasible with respect to $\hat{G}$.

If $\mathscr{H} \neq \emptyset$ and every graph in $\mathscr{H}$ pulls along F, then we say that $\hat{G}$ *forces to pull along* F. The notion of pulling along a face-sequence will now be extended to a family of face-sequences that all begin with the same face and subdivision vertex. Let $d \geq 2$, consider a small face f^* of G_0 and a subdivision vertex v_{e^*} that is on the boundary of f^*. Denote by u^* and w^* the two neighbors of v_{e^*} in G_0. For every $h \in [d]$, let $d_h \geq 1$ be an integer and let $F_h = (f_0^h, v_{e_1^h}, \ldots, v_{e_{d_h}^h}, f_{d_h}^h)$ be a face-sequence in G_0 with $f_0^h = f^*$ and $v_{e_1^h} = v_{e^*}$. Let $\hat{G}_0$ be the graph obtained from $\hat{G}$ by removing each vertex in $V(\hat{G}) \setminus V(G_0)$ that is embedded in a face f of G_0 with $f \in R(G_0) \setminus \{f_0^h, \ldots, f_{d_h}^h \colon h \in [d]\}$. Consider the set $\mathscr{H}$ of normal graphs $H' \subseteq \hat{G}_0$, for which the grid-path P, that contains u^* and w^*, contains neither v_{e^*} nor e^* and uses between u^* and w^* only vertices embedded in f^*, and for which every small face, except possibly the faces $f_{d_h}^h$ with $h \in [d]$, are feasible. If $\mathscr{H} \neq \emptyset$ and every graph in $\mathscr{H}$ pulls along F_h for each $h \in [d]$, then we say that $\hat{G}$ *forces to pull along* $(F_1, F_2, \ldots, F_d)$.

Next, some gadgets are defined, which will then be added to G_0 in order to obtain a graph G_ϕ such that (k, ℓ, G_ϕ) is an instance of the HGM Problem. Each gadget will require a certain number of faces, which will be modified and each modified face will be chosen such that it has boundary distance at least two. When a face f of G_0 has been chosen to be modified according to a gadget Y, then f is called a Y *face*. If $\hat{G}$ is a graph obtained from G_0 by adding a gadget Y, then the subgraph of $\hat{G}$ that is induced by all vertices embedded in Y faces is called the Y *subgraph*. The following figures always display a graph $\hat{G}$

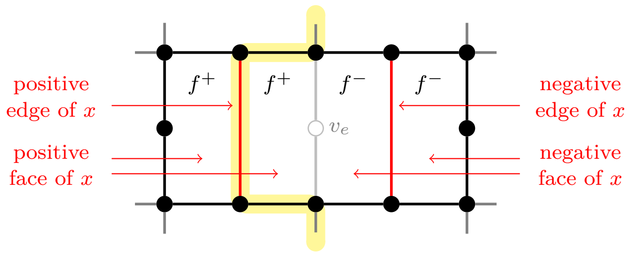


Figure 3.25: A decision gadget. The subdivision vertex v_e is deleted including its adjacent edges. A possible choice for a grid-path that is induced by the decision column of x is marked in yellow.

obtained from G_0 by adding a gadget. Subdivision vertices that have been replaced by their original edges are colored gray. New edges and vertices, i. e., edges and vertices that were inserted when constructing $\hat{G}$, are colored red, except for the edges that are inserted when replacing a subdivision vertex by its original edge. Vertices and edges that do not belong to the gadget itself are colored gray and orange depending on whether they are new. The modifications applied to G_0 often split a face of G_0 into several faces. In that case, each face of $\hat{G}$ that arises from a face f of G_0 is labeled with f.

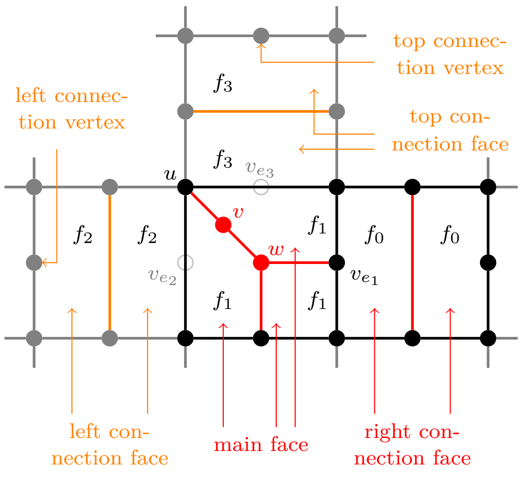
Decision and Bifurcation Gadgets

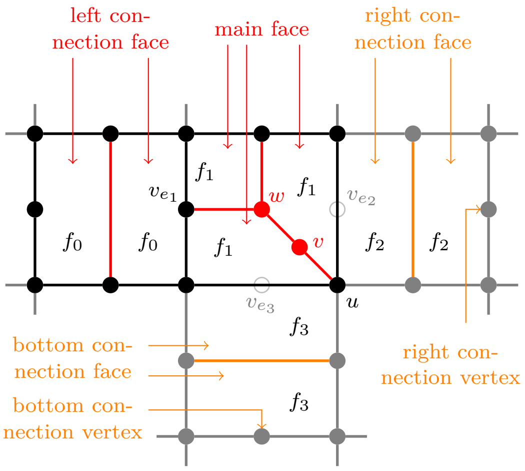
First, the *variable gadget* is presented, which consists of a decision gadget and several bifurcation gadgets. The *decision gadget* consists of two small faces of G_0 that are horizontally next to each other, such that the subdivision vertex in between them is deleted and both are split vertically, see Figure 3.25. More precisely, consider a variable $x \in \mathcal{U}$ and let v_e be a subdivision vertex of G_0 such that e is a vertical edge of $\tilde{G}_k$, i. e., $e \subseteq \tilde{C}_i$ for some $i \in [k-1] \setminus \{1\}$. Observe that v_e cannot be on the boundary of the large face of G_0. Let f^+ be the small face of G_0 whose right subdivision vertex is v_e and let f^- be the small face of G_0 whose left subdivision vertex is v_e. The faces f^+ and f^- are called the *positive face* and the *negative face* of the variable x, respectively. The decision gadget is to delete v_e including both edges incident to v_e and to split f^+ and f^- vertically. The edge inserted to split f^+ is referred to as the *positive edge of* x and the edge inserted to split f^- is referred to as the *negative edge of* x. Let $\hat{G}$ be the graph obtained from G_0 by adding a decision gadget in this way and consider a normal graph $H \subseteq \hat{G}$. The column of H that contains the two neighbors of v_e is called the *decision column* of x.

Claim 3.45.
Let $\hat{G}$ be a graph obtained from G_0 by adding a decision gadget corresponding to a variable $x \in \mathcal{U}$. Consider a normal subgraph $H \subseteq \hat{G}$, and let P be the grid-path induced by the decision column of x in H. Then P uses either the positive edge of x or the negative edge of x.

Proof. Let x, $\hat{G}$, H, and P be as in the statement. Denote by v_e the subdivision vertex of G_0 that was removed when constructing $\hat{G}$ and let $i, j \in [k]$ be integers such that $e = \{(i,j),(i,j+1)\}$. Observe that P cannot use a vertex in $V_{\text{forb}} = (\tilde{R}_j \cup \tilde{R}_{j+1}) \setminus \{(i,j),(i,j+1)\}$ as these vertices are used by the other columns of H since H is normal. In $\hat{G} - V_{\text{forb}}$ every path joining (i,j) to $(i,j+1)$ uses either the positive edge of x or the negative edge of x. Hence, P uses one of these edges. □

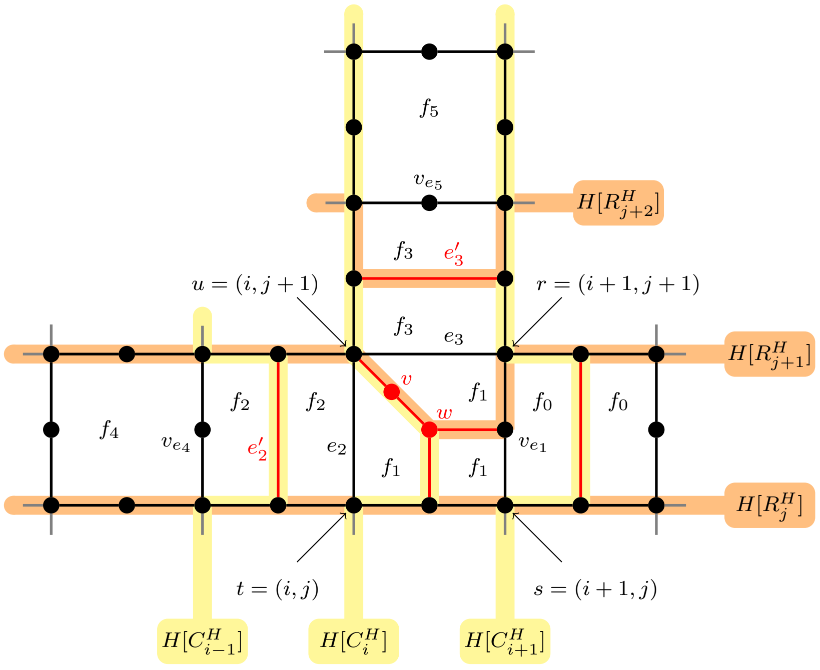
A *positive bifurcation gadget* consists of two faces of G_0, that are horizontally next to each other. In the left face, the left and the top subdivision vertex are replaced by their original edges and the right face is split vertically. Furthermore, two new vertices and four new edges are embedded in the left face





a) A positive bifurcation gadget. The non-solid subdivision vertices v_{e_2} and v_{e_3} are replaced by their original edges.

b) A negative bifurcation gadget. The non-solid subdivision vertices v_{e_2} and v_{e_3} are replaced by their original edges.



c) Proof of Claim 3.46a)

Figure 3.26: Bifurcation gadgets.

such that it is divided into three smaller faces, see Figure 3.26a). More precisely, let f_1 be a small face of G_0 with boundary distance at least 4. Let v_{e_2} and v_{e_3} be the left and the top subdivision vertex of f_1, respectively. For $h \in \{2, 3\}$, replace v_{e_h} by its original edge. Denote by u the vertex that the edges e_2 and e_3 have in common. Embed two new, adjacent vertices, called v and w, in the face f_1, add an edge joining v to u, an edge joining w to the bottom subdivision vertex of f_1, as well as an edge joining w to the right subdivision vertex of f_1. Furthermore, the face f_0 of G_0 whose left subdivision vertex is the right subdivision vertex of f_1 is split vertically. The face f_1 is called the *main face* of the positive bifurcation gadget and the face f_0 is called the *right connection face* of the positive bifurcation gadget. This finishes the description of the positive bifurcation gadget. It will be used in a way such that the face f_2 of G_0 whose right subdivision vertex is v_{e_2} is split vertically and the face f_3 of G_0 whose bottom subdivision vertex is v_{e_3} is split horizontally. The faces f_2 and f_3 are called the *left connection face* and the *top connection face* of the positive bifurcation gadget, but they do not belong to the positive bifurcation gadget. The left subdivision vertex of f_2 and the top subdivision vertex of f_3 are called the *left and the top connection vertex* of the positive bifurcation gadget, respectively. Note that both these vertices exist as otherwise f_2 or f_3 would have boundary distance 0, which contradicts that f_1 has boundary distance at least 4.

Claim 3.46.
Let $\hat{G}$ be the graph obtained from G_0 by adding one positive bifurcation gadget, splitting the left connection face of the positive bifurcation gadget vertically, and splitting the top connection face of the positive bifurcation gadget horizontally. Let F_1 be the face-sequence of G_0 that contains the right connection face, the main face, the left connection face, as well as the face whose right subdivision vertex is the left connection vertex in this order and no other faces of G_0. Moreover, let F_2 be the face-sequence of G_0 that contains the right connection face, the main face, the top connection face, as well as the face whose bottom subdivision vertex is the top connection vertex in this order and no other faces of G_0. Then, the following holds.

a) The graph $\hat{G}$ forces to pull along (F_1, F_2).
b) For every normal graph $H \subseteq \hat{G}$ that relaxes along the face-sequences F_1 and F_2, each face of H that corresponds to a face contained in F_1 or F_2 is feasible with respect to $\hat{G}$.

Proof.

a) Let $\hat{G}$, F_1, and F_2 be as in the statement, and let u, v, w, $f_0, \ldots, f_3$, v_{e_2}, and v_{e_3} be defined in the same way as when introducing the positive bifurcation gadget, see Figure 3.26c). Let $i, j \in [k]$ be two integers such that $u = (i, j+1)$. Moreover, define $r := (i+1, j+1)$, $s := (i+1, j)$, and $t := (i, j)$. Let v_{e_1} be the right subdivision vertex of f_1, let v_{e_4} and v_{e_5} be the left and the top connection vertex of the positive bifurcation gadget, respectively. Denote by f_4 the face of G_0 whose right subdivision vertex is v_{e_4} and denote by f_5 the face of G_0 whose bottom subdivision vertex is v_{e_5}. So $F_1 = (f_0, v_{e_1}, f_1, v_{e_2}, f_2, v_{e_4}, f_4)$ and $F_2 = (f_0, v_{e_1}, f_1, v_{e_3}, f_3, v_{e_5}, f_5)$. Let $H \subseteq \hat{G}$ be a normal graph such that each small face of H is feasible with respect to $\hat{G}$, except possibly the faces of H that correspond to f_4 and f_5, and such that the column C^H_{i+1} of H, i. e., the column containing r and s, does not contain v_{e_1} and such that, between r and s, the grid-path $H[C^H_{i+1}]$ uses only vertices embedded in f_0. In the following, the embedding of H that is induced by the embedding of the plane graph $\hat{G}$ is considered. This is not a restriction as H is a minimal graph containing a $k \times k$ grid as a minor and, hence, H is uniquely embeddable due to Remark 3.15.

First, observe that, for each $h \in \{0, 1, 2\}$, Claim 3.44b) implies that the grid-path $H[R^H_j]$ uses the bottom subdivision vertex $v_{e'}$ of f_h as the face whose top subdivision vertex is $v_{e'}$ has boundary distance at least 2 and has not been modified when constructing $\hat{G}$. As f_4 and the face whose top subdivision vertex is the bottom subdivision vertex of f_4 both have not been modified when constructing $\hat{G}$, Claim 3.44a) implies that the grid-path $H[R_j^H]$ uses the bottom subdivision vertex of f_4. So, roughly speaking, $H[R_j^H]$ does not make a detour between $(i-2, j)$ and $(i+2, j)$. Similarly, it follows that the grid-path $H[R_{j+1}^H]$ uses the top subdivision vertex of f_h for all $h \in \{0, 2, 4\}$, the grid-path $H[C_i^H]$ uses the left subdivision vertex of f_h for all $h \in \{3, 5\}$, and the grid-path $H[C_{i+1}^H]$ uses the right subdivision vertex of f_h for all $h \in \{3, 5\}$.

As C_{i+1}^H cannot contain any grid vertices that are on the boundary of f_0 except r and s, the grid-path $H[C_{i+1}^H]$ uses the top and bottom subdivision vertex of f_0 as well as the edge joining them. So, the vertices r and s, as well as the top and the bottom subdivision vertex of f_0 are on the boundary of the face f_1' of H that corresponds to the face f_1. As argued before, the grid-path $H[R_j^H]$ uses only the bottom subdivision vertex of f_1 between t and s. For a contradiction assume that the grid-path $H[R_{j+1}^H]$ uses e_3. Then the face f_1' of H would contain the vertices u, v, w, and v_{e_1} as well as the bottom subdivision vertex of f_1, as C_i^H contains neither r nor s. This implies that f_1' would contain at least 9 vertices of $\hat{G}$ and, hence, f_1' would not be feasible with respect to $\hat{G}$. So, the grid-path $H[R_{j+1}^H]$ uses (u, v, w, v_{e_1}, r) as a subpath to join u to r. Similarly, it follows that the grid-path $H[C_i^H]$ does not use e_2 and contains v, w, and the bottom subdivision vertex of f_1.

The face f_2' of H that corresponds to f_2 contains u, v, w, t, and the bottom subdivision vertex of f_1 in its boundary. So, in order for f_2' to be feasible with respect to $\hat{G}$, the graph H contains the edge e_2' that was inserted to split f_2 vertically. As H is a minimal graph containing a $k \times k$ grid as a minor, the edge e_2' is used by the grid-path $H[C_{i-1}^H]$. Consequently, H pulls along the face-sequence F_1. Similarly, one can argue that $H[R_{j+2}^H]$ does not use v_{e_5} and contains the edge e_3' that was inserted to split f_3 horizontally. Hence, H pulls along F_2. Recall that the faces of H that correspond to f_4 and f_5 do not need to be feasible with respect to $\hat{G}$. Now, it is easy to see that such a graph H indeed exists and therefore $\hat{G}$ forces to pull along (F_1, F_2).

b) Using the same notation as in the proof of Part a), it is easy to verify that the faces f_h with $h \in \{1, 2, 3\}$ are feasible with respect to $\hat{G}$. Claim 3.44c) implies that the top and bottom subdivision vertex of f_0 are used by the grid-paths $H[R_{j+1}^H]$ and $H[R_j^H]$, respectively and, hence, the face of H that corresponds to f_0 is feasible with respect to $\hat{G}$. Similarly, one argues that the faces f_4 and f_5 are feasible with respect to $\hat{G}$. □

There is also a *negative bifurcation gadget*, which is obtained by turning the positive bifurcation gadget around 180 degrees, see Figure 3.26b). It consists of the main face and the left connection face and defines also a right and a bottom connection face, that do not belong to the negative bifurcation gadget. The negative bifurcation gadget will be used in a way such that the right connection face is split vertically and the bottom connection face is split horizontally. Furthermore, it defines a right and a bottom connection vertex. Analog to Claim 3.46 the following is derived.

Claim 3.47.
Let $\hat{G}$ be the graph obtained from G_0 by adding one negative bifurcation gadget, splitting the right connection face of the negative bifurcation gadget vertically, and splitting the bottom connection face of the negative bifurcation gadget horizontally. Let F_1 be the face-sequence of G_0 that contains the left connection face, the main face, the right connection face, as well as the face whose left subdivision vertex is the right connection vertex in this order and no other faces of G_0. Moreover, let F_2 be the face-sequence of G_0 that contains the left connection face, the main face, the bottom connection face, as well as the face whose

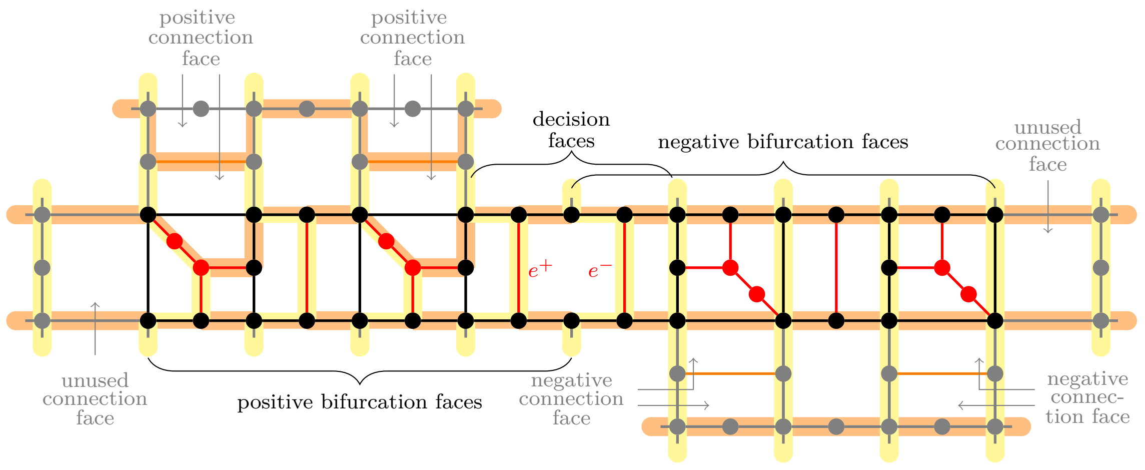


Figure 3.27: The extended variable gadget. The negative subgraph is highlighted. The labeled faces all refer to faces of G_0, the positive edge is labeled e^+ and the negative edge is labeled e^-.

top subdivision vertex is the bottom connection vertex in this order and no other faces of G_0. Then, the following holds.

a) The graph $\hat{G}$ forces to pull along (F_1, F_2).

b) For every normal graph $H \subseteq \hat{G}$ that relaxes along the face-sequences F_1 and F_2, each face of H that corresponds to a face contained in F_1 or F_2 is feasible with respect to $\hat{G}$.

Assembling the Variable Gadget

Consider a variable $x \in \mathcal{U}$. The variable gadget for x consists of $2\deg^+(x)$ positive bifurcation faces and $2\deg^-(x)$ negative bifurcation faces, such that one positive and one negative bifurcation face are also decision faces and the faces are arranged horizontally next to each other, as in Figure 3.27. More precisely, let $(f_1, v_{e_2}, \dots, v_{e_d}, f_d)$ be a face-sequence with $d := 2(\deg^+(x) + \deg^-(x))$ such that v_{e_h} is the right subdivision vertex of f_{h-1} and the left subdivision vertex of f_h for all $h \in [d] \setminus \{1\}$. Define $h_x := 2\deg^+(x)$. For each $h \in [\deg^+(x)]$, the faces f_{2h-1} and f_{2h} are modified according to the positive bifurcation gadget. For each integer h with $\deg^+(x) + 1 \leq h \leq \deg^+(x) + \deg^-(x)$, the faces f_{2h-1} and f_{2h} are modified according to the negative bifurcation gadget, and the subdivision vertex $v_{e_{h_x+1}}$ and its incident edges are removed. Then the faces f_{h_x} and f_{h_x+1} are automatically modified according to the decision gadget and are considered to be decision faces as well as bifurcation faces. Recall that the decision gadget defines a decision column for the variable x as well as a positive and a negative edge of x. Each top connection face of a positive bifurcation gadget is called a *positive connection face* of the variable gadget and each bottom connection face of a negative bifurcation gadget is called a *negative connection face* of the variable gadget. Furthermore, each top connection vertex of a positive bifurcation face is called a *positive connection vertex* of the extended variable gadget and each bottom connection vertex of a negative bifurcation face is called a *negative connection vertex* of the extended variable gadget. Moreover, the left connection face of the positive bifurcation gadget placed in the faces f_1 and f_2 is called the *unused left connection face* and the right connection face of the bifurcation gadget placed in the faces f_{d-1} and f_d is called the *unused right connection face* of the variable gadget. So, the variable gadget for x defines $\deg^+(x)$ positive connection faces, $\deg^-(x)$ negative connection faces, and two unused connection faces, such that none of these connection faces belongs to the variable gadget. The *extended variable gadget* consists of the variable gadget and all its connection faces, with the following modifications. Each top connection face is split horizontally, each bottom connection face is split horizontally, and in each unused connection face, the bottom and the top subdivision vertices are replaced by their original edges.

Some properties of the extended variable gadget are stated in the next claim. It uses the following definitions. Consider a graph $\hat{G}$ obtained from G_0 by adding one extended variable gadget corresponding to a Boolean variable $x \in \mathcal{U}$ such that each extended variable face has boundary distance at least 2. Denote by f^+ and f^- the positive and the negative face of the decision gadget that is part of the variable gadget corresponding to x. A normal graph $H \subseteq \hat{G}$ is *negative at the variable gadget* corresponding to x if it has the following properties

- the decision column of x uses the negative edge of x,
- for each positive connection face f as well as the unused left connection face f, the subgraph H is pulling along the f^+,f-face-sequence, whose internal faces all are positive bifurcation faces, and
- for each negative connection face f as well as the unused right connection face f, the subgraph H relaxes along the f^-,f-face-sequence, whose internal faces all are negative bifurcation faces,

see Figure 3.27. It is easy to check that such a subgraph H exists. Consider a normal subgraph $H \subseteq \hat{G}$ that is negative at the variable gadget corresponding to x. Observe that, for each positive connection vertex v_e, one of the faces of H, that corresponds to the face of G_0 whose bottom or top subdivision vertex is v_e, is not feasible with respect to $\hat{G}$. Nevertheless, the face of H that corresponds to the unused left connection face is feasible with respect to $\hat{G}$ due to the modifications applied to it. Similarly, a normal graph $H \subseteq \hat{G}$ is *positive at the variable gadget* corresponding to x if it has the following properties

- the decision column of x uses the positive edge of x,
- for each positive connection face f and the unused left connection face f, the subgraph H relaxes along the f^+,f-face-sequence, whose internal faces all are positive bifurcation faces, and
- for each negative connection face f and the unused right connection face f, the subgraph H is pulling along the f^-,f-face-sequence, whose internal faces all are negative bifurcation faces.

Claim 3.48.

Let $\hat{G}$ be a graph obtained from G_0 by adding one extended variable gadget corresponding to a Boolean variable x such that each extended variable face has boundary distance at least 2. Denote by f^+ and f^- the positive and the negative face of the decision gadget. Let $\mathcal{F}^+$ be the family of face-sequences that, for each positive connection face f of the considered variable gadget, contains the f^-,f-face-sequence, whose internal faces all belong to the variable gadget, and let $\mathcal{F}^-$ be the family of face-sequences that, for each negative connection face f of the considered variable gadget, contains the f^+,f-face-sequence, whose internal faces all belong to the variable gadget.

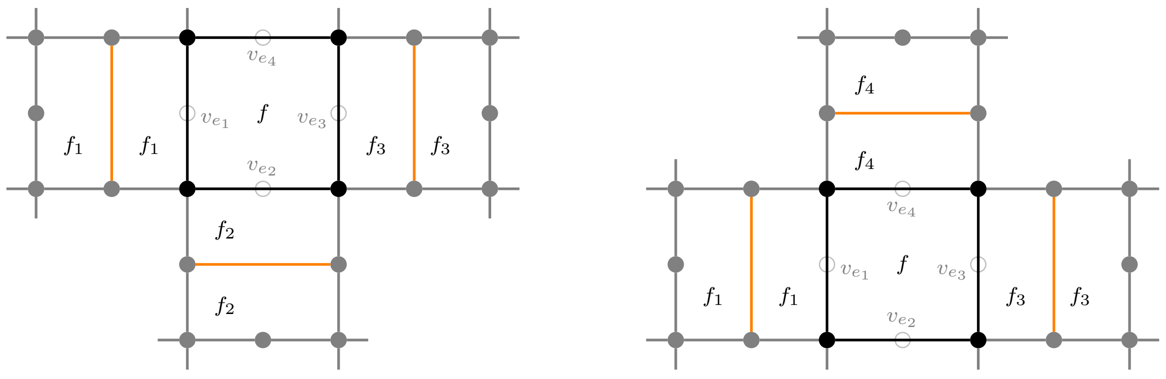
a) The graph $\hat{G}$ forces to pull along $\mathcal{F}^+$.

b) The graph $\hat{G}$ forces to pull along $\mathcal{F}^-$.

c) For every normal graph $H \subseteq \hat{G}$, that is negative at the variable gadget, each face of H corresponding to a variable face, a negative connection face, or an unused connection face of the extended variable gadget is feasible with respect to $\hat{G}$.

d) For every normal graph $H \subseteq \hat{G}$, that is positive at the variable gadget, each face of H corresponding to a variable face, a positive connection face, or an unused connection face of the extended variable gadget is feasible with respect to $\hat{G}$.

e) For every normal graph $H \subseteq \hat{G}$, the grid-path of H induced by the decision column of x uses either the positive edge of x or the negative edge of x.



a) An extended positive clause gadget. **b)** An extended negative clause gadget.

Figure 3.28: The clause gadget. The gray subdivision vertices $v_{e_1}, \dots, v_{e_4}$ are replaced by their original edges.

Proof.

a) Follows from Claim 3.46a).
b) Follows from Claim 3.47a).
c) Easy to check.
d) Easy to check.
e) Analogously to the proof of Claim 3.45. □

Clause Gadgets

A *clause gadget* consists of one small face f of G_0 with boundary distance at least 4 such that each subdivision vertex on the boundary of f is replaced by its original edge. The clause gadget will be used in the following way. Let f be a face of G_0 with boundary distance at least 2 and denote by v_{e_1}, v_{e_2}, v_{e_3}, and v_{e_4} its left, bottom, right, and top subdivision vertex, respectively. For $h \in [4]$, let f_h be the face of G_0 such that v_{e_h} is on the boundary of f_h and $f_h \neq f$. Place a clause gadget in f, i. e., replace v_{e_h} by its original edge e_h for all $h \in [4]$. To obtain the *extended positive clause gadget*, which consists of f, f_1, f_2, and f_3, split f_1 vertically, split f_2 horizontally, and split f_3 vertically, see Figure 3.28a). To obtain the *extended negative clause gadget*, which consists of f, f_1, f_3, and f_4, split f_1 vertically, split f_3 vertically, and split f_4 horizontally, see Figure 3.28b). Observe that the extended negative clause gadget is also obtained by rotating the extended positive clause gadget by 180 degrees. The following claim states some properties of the extended clause gadget, which is either a positive extended clause gadget or a negative extended clause gadget.

Claim 3.49.
Let $\hat{G}$ be a graph obtained by adding an extended clause gadget such that the clause face has boundary distance at least 4. Denote by f the clause face and denote by f_1, f_2, and f_3 the other extended clause faces. For $h \in [3]$, let v_{e_h} be the unique subdivision vertex of G_0 that is on the boundary of f and on the boundary of f_h.

a) In every normal graph $H \subseteq \hat{G}$, that pulls along (f_h, v_{e_h}, f) for all $h \in [3]$, the face of H that corresponds to f is not feasible with respect to $\hat{G}$.
b) In every normal graph $H \subseteq \hat{G}$, for which there is an $h \in [3]$ such that H relaxes along (f_h, v_{e_h}, f), the face of H that corresponds to f is feasible with respect to $\hat{G}$.

Proof.

a) Easy to check.

b) Let $\hat{G}$, as well as f_h and v_{e_h} for $h \in [3]$ be as in the statement. Let v_{e_4} be the fourth subdivision vertex on the boundary of f and denote by f_4 the face of G_0 that contains v_{e_4} on its boundary and is not f. Consider a normal subgraph $H \subseteq \hat{G}$ that relaxes along (f_1, v_{e_1}, f), the other two cases are analogous. For $h \in [4]$, let u_h and w_h be the neighbors of v_{e_h} in G_0. The grid-path of H that joins u_1 to w_1 uses the edge e_1 as H relaxes along (f_1, v_{e_1}, f) and, as f_4 and f are faces of $\hat{G}$, Claim 3.44a) implies that the grid-path of H that contains u_4 and w_4 uses e_4. For $h \in \{2, 3\}$, to join u_h to w_h the grid-path of H that contains u_h and w_h can either use the edge e_h or the two subdivision vertices on the boundary of f_h that were joined with a new edge when splitting f_h. Hence, the face f' of H that corresponds to f contains at most four grid vertices plus at most 4 subdivision vertices from the boundaries of f_2 and f_3. Therefore, f' is feasible with respect to $\hat{G}$. □

Wire Gadgets

Before presenting the last gadget, some further notation is introduced. Let f be a face of G_0 with boundary distance at least 2. For two distinct subdivision vertices v_e and $v_{e'}$ on the boundary of f, we say that v_e and $v_{e'}$ are *on opposite sides of* f if $e \cap e' = \emptyset$, i. e., v_e and $v_{e'}$ are the bottom and the top subdivision vertex of f or v_e and $v_{e'}$ are the left and the right subdivision vertex of f. Let $F = (f_0, v_{e_1}, f_1, \ldots, v_{e_d}, f_d)$ be a face-sequence for some integer d. Then, F is called *straight* if $d \leq 1$ or, for all $h \in [d-1]$, the subdivision vertices v_{e_d} and $v_{e_{d+1}}$ are on opposite sides of f_d. Moreover, for an integer $h \in [d-1]$, the face-sequence F *is straight except for a turn in* f_h if F is not straight but $F' := (f_0, v_{e_1}, \ldots, f_h)$ and $F'' := (f_h, v_{e_{h+1}}, \ldots, f_d)$ are straight.

A *straight wire gadget* consists of a straight face-sequence F such that each face in F, except the last face of F, is split orthogonal to the direction of F, see Figure 3.29a). More precisely, consider a straight face-sequence $F = (f_0, v_{e_1}, f_1, \ldots, f_d)$ with $d \geq 1$ along which we want to insert a straight wire gadget. If e_h is a vertical edge of $\tilde{G}_k$ for all $h \in [d]$, then, for every $h \in [d]$ the face f_{h-1} is split vertically. Otherwise, e_h is a horizontal edge of $\tilde{G}_k$ for all $h \in [d]$ and, for all $h \in [d]$, the face f_{h-1} is split horizontally. Furthermore, a *corner wire gadget* consists of a face-sequence F that is straight except for one turn at a face f and where the two straight parts of F are modified according to the straight wire gadget, see Figure 3.29c). More precisely, consider a face-sequence $F = (f_0, v_{e_1}, \ldots, f_d)$ with $d \geq 2$ that is straight except for a turn in f_h for some $h \in [d-1]$. Then, inserting a corner wire gadget along F means to modify the face-sequences $(f_0, v_{e_1}, \ldots, f_h)$ and $(f_h, v_{e_{h+1}}, \ldots, f_d)$ according to the straight wire gadget.

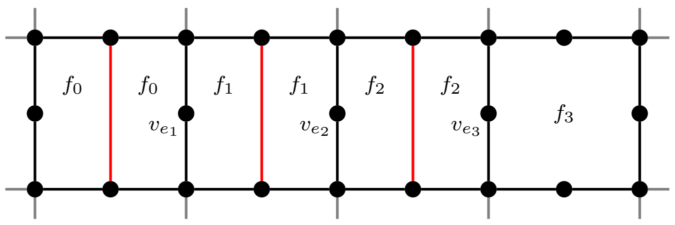
Claim 3.50.
Let F be a face-sequence in G_0 that is either straight or straight except for a turn and such that each face of f has boundary distance at least 4. Consider the graph $\hat{G}$ that is obtained from G_0 by adding a wire gadget along F.

a) The graph $\hat{G}$ forces to pull along F.

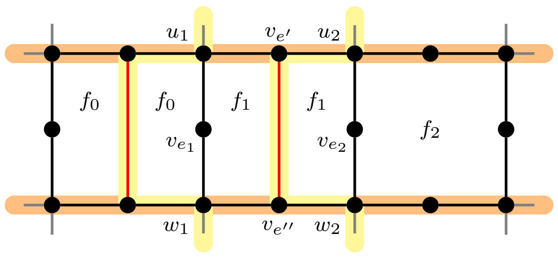
b) For every normal graph H that relaxes along F, each face of H that corresponds to a face in F is feasible with respect to $\hat{G}$.

Proof.

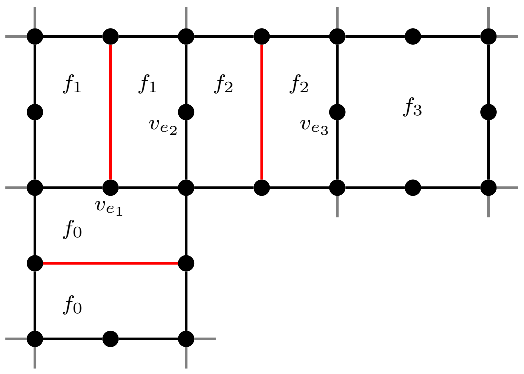
a) Let $F = (f_0, v_{e_1}, f_1, \ldots, f_d)$ and let $\hat{G}$ be as in the statement. First, assume that F is a straight face-sequence. If $d = 1$, there is nothing to show and as the following argument can be extended



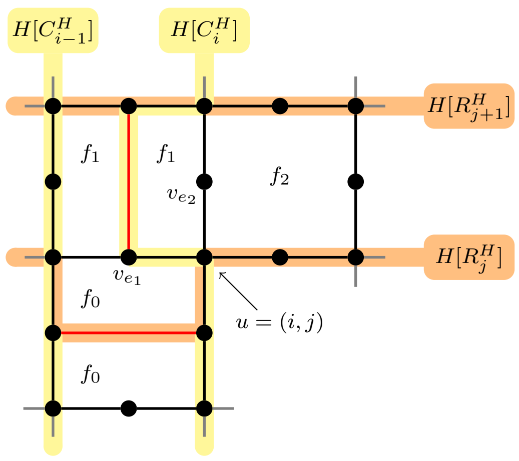
a) A straight wire gadget.



b) Proof of Claim 3.50a).



c) A corner wire gadget.



d) Proof of Claim 3.50a).

Figure 3.29: Wire Gadgets.

inductively it suffices to consider the case when $d = 2$. For $h \in [2]$, let u_h and w_h be the neighbors of v_{e_h} in G_0, such that u_1 and u_2 are adjacent in $\tilde{G}_k$ and w_1 and w_2 are adjacent in $\tilde{G}_k$, see Figure 3.29b). Consider a normal graph $H \subseteq \hat{G}$ such that the grid-path of H that contains u_1 and w_1 does not use v_{e_1} and uses between u_1 and w_1 only vertices embedded in f_0. Assume that every small face of H, except possibly the face of H that corresponds to f_2 is feasible with respect to $\hat{G}$. The grid-path of H that contains u_1 and w_1 uses the new edge that was inserted to split f_0 when constructing $\hat{G}$ from G_0 as it cannot use any grid vertices that are on the boundary of f_0 other than u_1 and w_1. Denote by $v_{e'}$ and $v_{e''}$ the two subdivision vertices of G_0 whose neighborhood is $\{u_1, u_2\}$ and $\{w_1, w_2\}$, respectively. Claim 3.44c) implies that the grid-path of H that contains u_1 and u_2 uses $v_{e'}$ as well as that the grid-path of H that contains w_1 and w_2 uses $v_{e''}$. So, the face of H that corresponds to f_1 can only be feasible with respect to $\hat{G}$ if the grid-path of H that contains u_2 and w_2 uses $v_{e'}$ and $v_{e''}$ as well as the edge $\{v_{e'}, v_{e''}\}$. Consequently, $\hat{G}$ forces to pull along F.

Assume now that F is straight except for one turn in f_h with $h \in [d-1]$. As the argument above implies that $\hat{G}$ forces to pull along $(f_0, v_{e_1}, \ldots, f_h)$ and $(f_h, v_{e_{h+1}}, \ldots, f_d)$, it suffices to consider the case when $d = 2$ and F makes a turn in f_1. Assume that v_{e_1} is the bottom subdivision vertex of f_1 and that v_{e_2} is the right subdivision vertex of f_1, see Figure 3.29d), all other cases are analogous. Denote by $u = (i, j)$ the common neighbor of v_{e_1} and v_{e_2}. Consider a normal graph $H \subseteq \hat{G}$ such that every face of H is feasible with respect to $\hat{G}$, except possibly the face of H that corresponds to f_2, and assume that the grid-path $H[R^H_j]$ does not use v_{e_1} and uses between $(i-1, j)$ and u only vertices embedded in f_0. Then, the grid-path $H[R^H_j]$ uses the edge inserted to subdivide f_0 horizontally. Furthermore, Claim 3.44b) implies that the grid-path $H[C^H_{i-1}]$ uses the left subdivision vertices of f_0 and f_1 as well as that the grid-path $H[R^H_{j+1}]$ uses the top subdivision vertex of f_1. So, in order for the face of H that corresponds to f_1 to be feasible with respect to $\hat{G}$, the grid-path $H[C^H_i]$ must use the edge inserted to split f_1 vertically. Consequently, $\hat{G}$ forces to pull along F.

b) Let $F = (f_0, v_{e_1}, f_1, \ldots, f_d)$, $\hat{G}$ and H be as in the statement. Clearly, the faces of H that correspond to a face f_h with $h \in [d-1]$ are feasible with respect to $\hat{G}$. Using Claim 3.44c) it follows that f_0 and f_d are feasible with respect to $\hat{G}$. □

Assembling the Graph G_ϕ

Next, some gadgets will be added to G_0 in order to construct a graph G_ϕ such that (k, ℓ, G_ϕ) is an instance of the HGM Problem. The variable gadgets for the variables in $\mathcal{U}$ will be added one after another to the middle row of G_0, such that there are two faces between two consecutive variable gadgets that will be modified according to the unused connection faces. More precisely, let $j_v = \lfloor \frac{1}{2} k \rfloor$ and let $F_{\mathcal{U}} := (f_1, v_{e_2}, f_2, \ldots, f_{k-1})$ be the straight face-sequence, where f_1 is the small face of G_0 whose boundary contains the grid vertices $(1, j_v)$ and $(1, j_v + 1)$ and f_{k-1} is the small face of G_0 whose boundary contains the grid vertices (k, j_v) and $(k, j_v + 1)$. Let $\mathcal{U} = \{x_1, \ldots, x_n\}$ and assume that $x_1, \ldots, x_n$ is the order in which the variables appear in $\mathcal{R}$. For $h \in [n]$, define $d_h := 2(\deg^+(x_h) + \deg^-(x_h))$. Furthermore, define $\hat{d}_1 := 2$ and, for $h \in [n] \setminus \{1\}$, define $\hat{d}_h := \hat{d}_{h-1} + d_{h-1} + 2$. Then, for each $h \in [n]$ the variable gadget for x_h is placed in the faces $f_{\hat{d}_h+1}, \ldots, f_{\hat{d}_h+d_h}$ and the faces $f_{\hat{d}_h}$ and $f_{\hat{d}_h+d_h+1}$ are modified according to the unused left connection face and the unused right connection face of the extended variable gadget for x_h, respectively. For each integer h with $2 \le h \le \hat{d}_n + d_n + 1$, the face f_h of $F_{\mathcal{U}}$ belongs to exactly one extended variable gadget. Furthermore,

$$\begin{aligned}\hat{d}_n + d_n + 1 &= \hat{d}_1 + \sum_{h \in [n-1]} (d_h + 2) + d_n + 1 = 2 + \sum_{h \in [n]} d_h + 2(n-1) + 1 \\ &= \sum_{h \in [n]} 2(\deg^+(x_h) + \deg^-(x_h)) + 2n + 1 = 6m + 2n + 1,\end{aligned}$$

as $\sum_{h \in [n]} \left(\deg^+(x_h) + \deg^-(x_h)\right)$ counts each vertical line of $\mathcal{R}$ once and each clause in $\mathcal{R}$ touches exactly three vertical lines. Recall that $k := 6m + 2n + 3$. So, $F_{\mathcal{U}}$ contains enough faces to place all variable gadgets next to each other with two unused connection faces in between consecutive variable gadgets and the face f_{k-1} is not modified. Denote by G_1 the graph obtained from G_0 by adding the variable gadgets and modifying their unused connection faces in the described way. Observe that each face in $F_{\mathcal{U}}$ except the two faces with boundary distance 0 are modified and each face that was modified when constructing G_1 from G_0 belongs to $F_{\mathcal{U}}$.

In the following, more faces of G_0 are chosen for placing clause and wire gadgets. When doing so, the faces refer to the graph G_0 and we assume that some of the faces of G_0 are already reserved but not modified according to the variable gadgets. Recall from Section 3.3.3 that two face-sequences F and F' are called *disjoint* if each face that is in F is not in F' and vice versa, as well as that two face-sequences F and F' are called *disjoint except for the last face* if F and F' are disjoint or the only violation to being disjoint is that the last face used in F is the last face used in F'. Using $\mathcal{R}$, one can choose a face f_C of G_0 for each $C \in \mathcal{C}$ and a face-sequence $\hat{F}_L$ in G_0 for each vertical line L in $\mathcal{R}$ such that the following is satisfied.

(i) For all $C \in \mathcal{C}$ the face f_C has boundary distance at least 4 and there is no variable $x \in \mathcal{U}$ such that f_C belongs to the extended variable gadget corresponding to x.

(ii) For all distinct $C, C' \in \mathcal{C}$ the faces f_C and $f_{C'}$ are distinct.

(iii) For each vertical line L in $\mathcal{R}$, the face-sequence $\hat{F}_L$ is either straight or straight except for one turn.

(iv) For each vertical line L in $\mathcal{R}$ that joins a variable $x \in \mathcal{U}$ to a positive clause $C \in \mathcal{C}$, the face-sequence $\hat{F}_L = (f_0, v_{e_1}, f_1, \ldots, f_d)$ satisfies the following: the face f_0 is a positive connection face of the variable gadget corresponding to x, the subdivision vertex v_{e_1} is the positive connection vertex that is on the boundary of f_0, and the face f_d is the face f_C.

(v) For each vertical line L in $\mathcal{R}$ that joins a variable $x \in \mathcal{U}$ to a negative clause $C \in \mathcal{C}$, the face-sequence $\hat{F}_L = (f_0, v_{e_1}, f_1, \ldots, f_d)$ satisfies the following: the face f_0 is a negative connection face of the variable gadget corresponding to x, the subdivision vertex v_{e_1} is the negative connection vertex that is on the boundary of f_0, and the face f_d is the face f_C.

(vi) For every grid vertex v in G_0, for which there are two distinct lines L and L' in $\mathcal{R}$ such that v is on the boundary of a face in $\hat{F}_L$ and v is on the boundary of a face in $\hat{F}_{L'}$, there is a clause C such that v is on the boundary of f_C and such that L and L' both touch the clause C in $\mathcal{R}$.

Indeed, $j_v = \left\lfloor \frac{1}{2}k \right\rfloor \geq 3m + n + 1$ and, hence, there is enough space for each clause $C \in \mathcal{C}$. The next claim states some additional properties of the face-sequences $\hat{F}_L$, where L is a vertical line in $\mathcal{R}$.

Claim 3.51.

a) *For each clause $C \in \mathcal{C}$, there are exactly three face-sequences $\hat{F}_L$ where L is a vertical line in $\mathcal{R}$ such that the last face of $\hat{F}_L$ is f_C.*

b) *For all distinct vertical lines L, L' in $\mathcal{R}$, the face-sequences $\hat{F}_L$ and $\hat{F}_{L'}$ are disjoint, except when L and L' touch the same clause in $\mathcal{R}$, in which case $\hat{F}_L$ and $\hat{F}_{L'}$ are disjoint except for the last face.*

c) *For each variable $x \in \mathcal{U}$ and for each connection face f of the extended variable gadget corresponding to x that is not an unused connection face, there is a face-sequence $\hat{F}_L$ with L in $\mathcal{R}$ such that f is the first face of $\hat{F}_L$.*

d) *For each vertical line L in $\mathcal{R}$, each face in $\hat{F}_L = (f_0, v_{e_1}, f_1, \ldots, f_d)$ has boundary distance at least 4 and, if $\hat{F}_L$ is not a straight face-sequence, then $\hat{F}_L$ makes a turn at f_h with $h \leq d - 2$.*

e) *For every grid vertex v, for which there is a variable $x \in \mathcal{U}$ and a line L in $\mathcal{R}$ such that v is on the boundary of a face belonging to the variable gadget corresponding to x and v is on the boundary of a face f in $\hat{F}_L$, the face f is the first face of $\hat{F}_L$ and f is a connection face of the variable gadget corresponding to x. In particular, for each line L in $\mathcal{R}$, and for each face f in the face-sequence $F_{\mathcal{U}}$, the face-sequence $\hat{F}_L$ does not use f.*

Proof.

a) This follows from the assumption that each clause $C \in \mathcal{C}$ contains exactly three variables.

b) Follows from (vi).

c) Part b) implies that, for distinct vertical lines L and L' of $\mathcal{R}$, the face-sequences $\hat{F}_L$ and $\hat{F}_{L'}$ start in different faces. Then the statement follows as, for each variable $x \in \mathcal{U}$, the variable gadget corresponding to x defines $\deg^+(x)$ positive connection faces and $\deg^-(x)$ negative connection faces.

d) By construction, each connection face defined by a variable gadget corresponding to a variable in $\mathcal{U}$ has boundary distance at least 4. Moreover, for each $C \in \mathcal{C}$, the face f_C has boundary distance at least 4 due to (i). Now the first part of the statement follows from (iii). If the second part of the statement was not true, then (vi) was violated.

e) For each vertical line L in $\mathcal{R}$, the face-sequence $\hat{F}_L = (f_0, v_{e_1}, f_1, \dots, f_d)$ satisfies that f_0 is a connection face of a variable gadget corresponding to a variable in $\mathcal{U}$ and v_{e_1} is either a positive or a negative connection vertex that is on the boundary of f_0 due to (iv) and (v). By (iii), the face f_0 is the only face of $\hat{F}_L$ that contains a vertex v such that v is on the boundary of a face in the face-sequence $F_{\mathcal{U}}$. □

For each face f of G_0 for which there is a clause C such that $f = f_C$ or for which there is a vertical line L in $\mathcal{R}$ such that $\hat{F}_L$ uses f, the face f is also a face of G_1 and f has not been modified when constructing G_1 from G_0 due to Claim 3.51e), except when f is a positive connection face of a variable gadget and its bottom subdivision vertex has been deleted and when f is a negative connection face of a variable gadget and its top subdivision vertex has been deleted. So, the following construction is feasible. For each vertical line L in $\mathcal{R}$ such that $\hat{F}_L$ is straight, add a straight wire gadget along $\hat{F}_L$ to G_1 and, for each vertical line L in $\mathcal{R}$ such that $\hat{F}_L$ is straight except for one turn, add a corner wire gadget along $\hat{F}_L$ to G_1. Let G_2 be the graph obtained in this way. Claim 3.51b) implies that the graph G_2 is planar as each face is split at most one time when constructing G_2 from G_1. When constructing G_2 from G_1, for each clause $C \in \mathcal{C}$, the face f_C is not modified. Let G_ϕ be the graph obtained from G_2 by adding a clause gadget in the face f_C for all clauses $C \in \mathcal{C}$, see Figure 3.30 for an example. Observe that, in each face f of G_0 at most a constant number of vertices and edges have been added to construct the graph G_ϕ. Consequently, G_ϕ has size polynomial in k and also polynomial in $n + m$, which is a lower bound on the size of ϕ.

For each vertical line L in $\mathcal{R}$ that joins a variable $x \in \mathcal{U}$ to a positive clause $C \in \mathcal{C}$, let F_L be the face-sequence $(f_0, v_{e_1}, f_1, \dots, v_{e_d}, f_d)$ where $f_d = f_C$ and f_0 is the negative face of x, which is obtained from $\hat{F}_L$ by adding f_0 and positive bifurcation faces corresponding to the variable gadget of x. For a variable $x \in \mathcal{U}$, denote by $\mathcal{F}_x^+$ the set of face-sequences that contains F_L for each vertical line L in $\mathcal{R}$ that joins x to a positive clause in $\mathcal{C}$. Similarly, for each vertical line L in $\mathcal{R}$ that joins a variable $x \in \mathcal{U}$ to a negative clause $C \in \mathcal{C}$, let $F_L = (f_0, v_{e_1}, f_1, \dots, v_{e_d}, f_d)$ be the face-sequence where $f_d = f_C$ and f_0 is the positive face of x, which is obtained from $\hat{F}_L$ by adding f_0 and negative bifurcation faces corresponding to the variable gadget of x. For a variable $x \in \mathcal{U}$ denote by $\mathcal{F}_x^-$ the set of face-sequences that contains F_L for each vertical line L in $\mathcal{R}$ that joins x to a negative clause in $\mathcal{C}$. The next claims state some properties of G_ϕ.

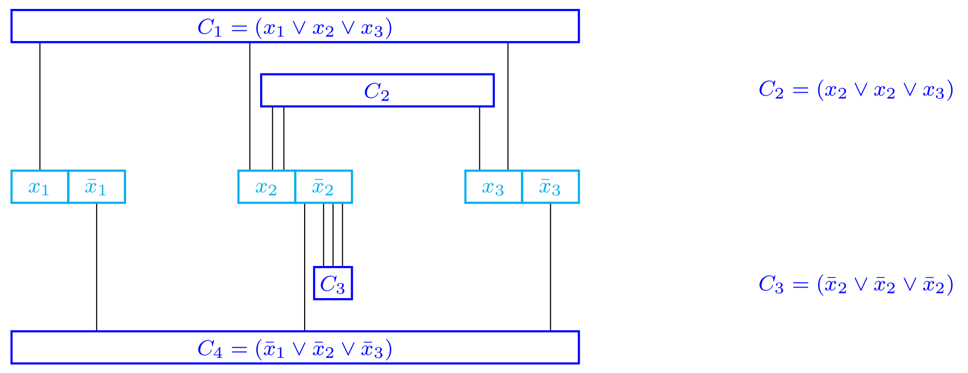
Claim 3.52.

a) Each face of G_0 that has been modified when constructing G_ϕ is either a variable face, an unused connection face, a wire face, or a clause face.

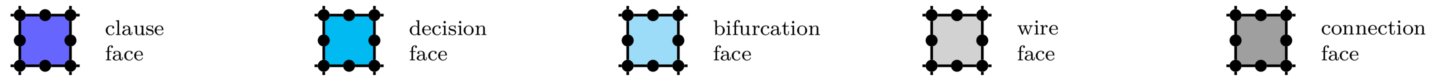
b) Each face of G_0 that has boundary distance zero has not been modified when constructing G_ϕ from G_0 and each face of G_0 that has boundary distance at most 2 has either not been modified or has been modified according to an unused connection face of an extended variable gadget.

c) For each $i \in [k]$, the set $\tilde{C}_i$ separates $\tilde{C}_1$ from $\tilde{C}_k$ in G_ϕ and, for each $j \in [k]$, the set $\tilde{R}_j$ separates $\tilde{R}_1$ from $\tilde{R}_k$ in G_ϕ.

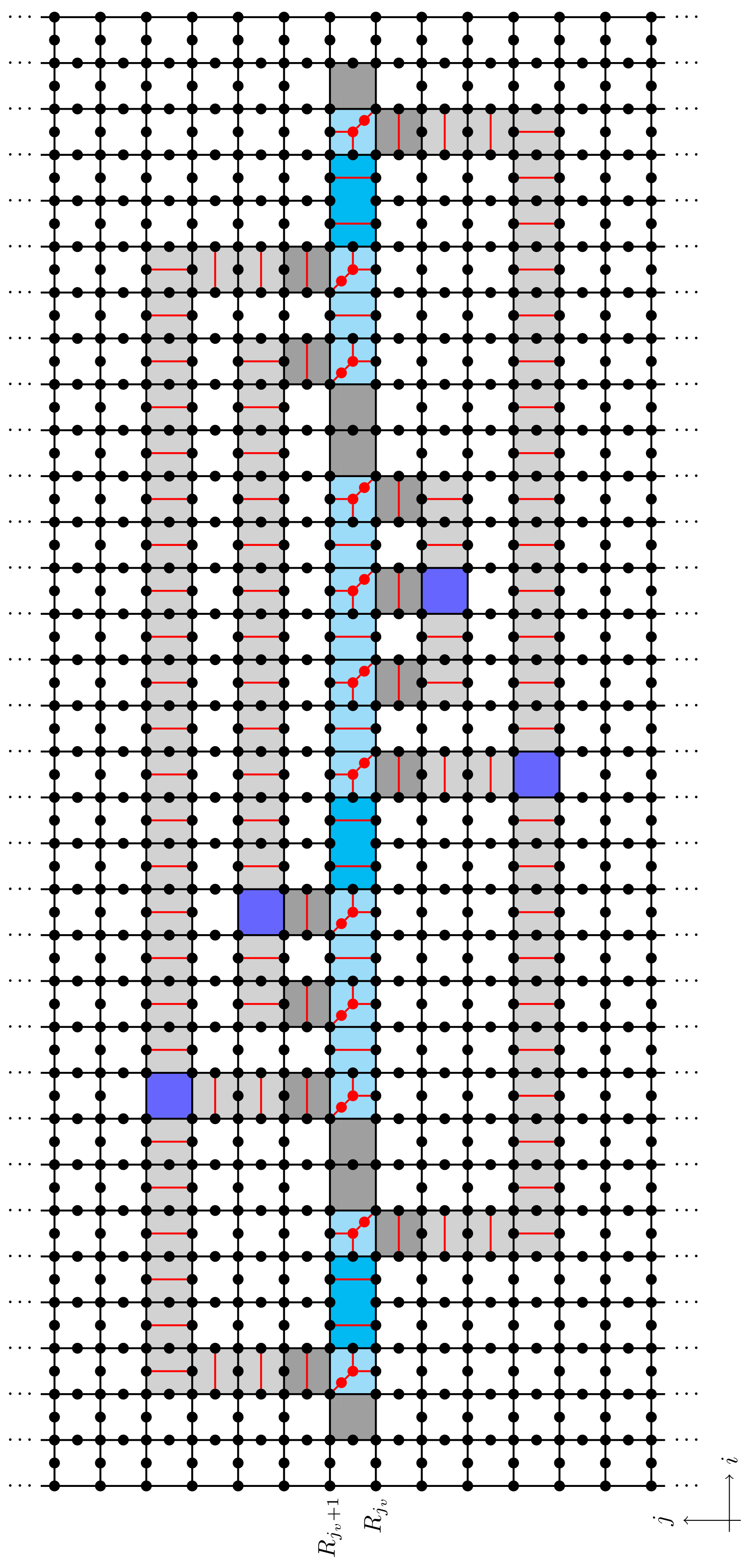
d) For each variable $x \in \mathcal{U}$, the graph G_ϕ forces to pull along the face-sequence $\mathcal{F}_x^+$ and the graph G_ϕ forces to pull along the face-sequence $\mathcal{F}_x^-$.


$C_1 = (x_1 \lor x_2 \lor x_3)$
C_2
x_1
$\bar{x}_1$
x_2
$\bar{x}_2$
x_3
$\bar{x}_3$
C_3
$C_4 = (\bar{x}_1 \lor \bar{x}_2 \lor \bar{x}_3)$
$C_2 = (x_2 \lor x_2 \lor x_3)$
$C_3 = (\bar{x}_2 \lor \bar{x}_2 \lor \bar{x}_2)$

a) Monotone rectilinear representation of ϕ.



b) Colors used in Part c). Observe that every decision face is also a bifurcation face and that every connection face that is not unused is also a wire face. Each positive and each negative connection face is split horizontally; unused connection faces are not split.



c) The graph G_ϕ. Recall that the vertex (i,j) for $i,j \in [k]$ is embedded at the coordinates (i,j), where the horizontal axis refers to the first coordinate and the vertical axis refers to the second coordinate.

Figure 3.30: Example of the graph G_ϕ for $\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_2 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$.

Proof.

a) Follows from Claim 3.51b) and e).

b) Follows from Claim 3.51d) and the construction of G_1.

c) Fix an $i \in [k-1] \setminus \{1\}$. Assume for a contradiction that $\tilde{C}_i$ does not separate $\tilde{C}_1$ from $\tilde{C}_k$ in G_ϕ. So there is a $\tilde{C}_1$,$\tilde{C}_k$-path P in $G_\phi - \tilde{C}_i$. Let $\hat{C}_i$ be the set of vertices that consists of $\tilde{C}_i$ and each subdivision vertex v_e of G_0 such that e is a vertical edge in column $\tilde{C}_i$ in $\tilde{G}_k$ and v_e is not replaced by its original edge when constructing G_ϕ from G_0. As $\hat{C}_i$ induces a path in G_ϕ that can be extended to a cycle by adding one new edge such that the graph remains plane and the sets $\tilde{C}_1$ and $\tilde{C}_k$ are on different sides of the cycle, each $\tilde{C}_1$,$\tilde{C}_k$-path in G_ϕ uses at least one vertex from $\hat{C}_i$. Therefore, P uses a subdivision vertex v_e such that e is a vertical edge in column $\tilde{C}_i$ in $\tilde{G}_k$. Let $e = \{(i,j),(i,j+1)\}$ and let f_1 and f_2 be the two faces of G_0 such that v_e is on the boundary of f_1 and f_2. Since P can use neither (i,j) nor $(i,j+1)$, there are two new edges e_1 and e_2 that were added to G_0 when constructing G_ϕ and such that e_i is embedded in f_i for $i \in [2]$. It is easy to see that neither f_1 nor f_2 can be a variable face, an unused decision face, or a clause face. So f_1 and f_2 are both wire faces. If f_1 and f_2 belong to the same face-sequence F_L for some vertical line in $\mathcal{R}$ then $e_1 \cap e_2 = \emptyset$. However, the faces f_1 and f_2 cannot belong to different face-sequences as otherwise (vi) was violated for $v = (i,j)$ or $v = (i,j+1)$. Consequently, there is no such path P and $\tilde{C}_i$ separates $\tilde{C}_1$ from $\tilde{C}_k$. Clearly the statement is true for $i = 1$ and $i = k$. For rows the argument is similarly, except that for showing that neither f_1 nor f_2 is a variable face, Claim 3.51e) is needed.

d) Follows from Claim 3.48a) and b), and Claim 3.50a) □

Claim 3.53.
The graph G_ϕ is uniquely embeddable.

Proof. Let G_0'' be the graph obtained from G_ϕ by deleting every vertex that is not in G_0 and also every edge that is neither in G_0 nor in $\tilde{G}_k$. So, roughly speaking, G_0' is a $k \times k$ grid where some edges are missing and some edges are subdivided. Let G_0' be the graph obtained from G_0'' by inserting the positive edge of x for each variable $x \in \mathcal{U}$. Then, G_0' is a minimal graph containing a $k \times k$ grid as a minor and Remark 3.15 implies that G_0' is uniquely embeddable. So, from now on, fix an embedding of G_0' and consider G_0' as a plane graph. Let $e = \{v_{e_1}, v_{e_2}\}$ be an edge that was inserted to split a small face f of G_0 when constructing G_ϕ. As f has boundary distance at least 2 by Claim 3.52b), f is the only face of G_0 with v_{e_1} and v_{e_2} on its boundary and it is easy to see that also in G_0' there is only one face f' such that v_{e_1} and v_{e_2} are both on the boundary of f'. Hence, there is only one way to add the edge e to the plane graph G_0'. As each face of G_0 is split at most once when constructing G_ϕ, this argument still applies when some of these edges have already been added to G_0'. So the plane graph G_1' obtained from G_0' by adding each edge, that when constructing G_ϕ was inserted into G_0 to split a face of G_0, is uniquely embeddable. The graph G_1' almost looks like G_ϕ, except for some modifications in the main bifurcation faces. Let f be a face of G_0 that has been modified according to a main bifurcation face. Denote by v and w the two new vertices that were added to f when modifying G_0 to obtain G_ϕ. In the following, a pair of such vertices (v, w) is called a *pair of new bifurcation vertices*. Denote by r, s, and t the three vertices on the boundary of f that are adjacent to v or w in G_ϕ. The graph G_0 only contains one face f_0 such that r, s, and t are on the boundary of f_0, namely $f_0 = f$, and similarly, also the graph G_1' contains only one face f_1 such that r, s, and t are on the boundary of f_1. Hence, there is only one way to add the vertices v and w to G_1' including their incident edges as in G_ϕ. As each face of G_0 that is modified according to a main bifurcation face is not modified due to another gadget, it follows that inserting one pair of new bifurcation vertices does not influence the possibilities of inserting another pair of new bifurcation vertices. Hence, the graph G'_2 obtained from G'_1 by inserting all pairs of new bifurcation vertices is uniquely embeddable. Observing that G'_2 is identical to G_ϕ shows that G_ϕ is uniquely embeddable. □

This completes the construction of G_ϕ and the statement of its properties. From now on, as G_ϕ is uniquely embeddable, consider G_ϕ to be a plane graph with this unique embedding.

The “if and only if” statement

After completing the construction, we now show that ϕ is a yes-instance of the Planar Monotone 3-SAT Problem if and only if (k, ℓ, G_ϕ) is a yes-instance of the HGM Problem, i. e., ϕ is satisfiable if and only if there is a feasible subgraph H of G_ϕ.

Claim 3.54.
If ϕ is satisfiable, then there is a feasible subgraph $H \subseteq G_\phi$.

Proof. Assume that ϕ is satisfiable. To prove the claim, a feasible, normal subgraph of G_ϕ is constructed. Let $T : \mathcal{U} \to \{\text{True}, \text{False}\}$ be a satisfying assignment of ϕ. The assignment of T for a variable x is called *good for a clause $C \in \mathcal{C}$* if C uses the variable x and either C is a positive clause and $T(x) = \text{True}$ or C is a negative clause and $T(x) = \text{False}$. Consider a plane graph $H \subseteq G_\phi$ with the following properties. The graph H contains every grid vertex of G_0 and the large face of H is bounded by B_{large} which was defined as the cycle that bounds the large face of G_0. This is possible as no grid vertex of G_0 was removed when constructing G_ϕ. So it remains to specify how each pair of grid vertices that are adjacent in $\tilde{G}_k$ are joined in H. Let $u = (i, j)$ and $u' = (i', j')$ be two grid vertices of G_0 that are adjacent in $\tilde{G}_k$ such that not both of them are on the boundary of the large face of G_0. Denote by v_e the subdivision vertex of G_0 whose neighbors are u and v. Let f and f' be the two small faces of G_0 with v_e on their boundary. If f or f' does not belong to a gadget or an extended gadget that was placed in G_0 when constructing G_ϕ, then H contains either v_e or e. So it remains to describe how two grid vertices u and u' that are adjacent in $\tilde{G}_k$ are joined when f and f' both belong to a gadget or an extended gadget. To do so, the following rules are applied.

- For each variable $x \in \mathcal{U}$ with $T(x) = \text{True}$, the subgraph H is positive at the variable gadget corresponding to x.
- For each variable $x \in \mathcal{U}$ with $T(x) = \text{False}$, the subgraph H is negative at the variable gadget corresponding to x.
- For each line L in $\mathcal{R}$, that joins a variable $x \in \mathcal{U}$ to some clause $C \in \mathcal{C}$, the graph H relaxes along $\hat{F}_L$ if the assignment of T for x is good for C and otherwise H pulls along $\hat{F}_L$.
- For each subdivision vertex of G_ϕ that is on the boundary of a left unused connection face as well as a right unused connection face, the graph H contains v_e as well as the two edges incident to v_e.

To see that H is a minimal graph containing a $k \times k$ grid as a minor, define columns and rows of H as follows. For $i \in [k-1] \setminus \{1\}$, let $C_i^H \subseteq V(H)$ be the set of vertices that contains each vertex in $\tilde{C}_i$ and the vertices added to H for joining two vertices in $\tilde{C}_i$ that are adjacent in $\tilde{G}_k$. Define $C_1^H := \tilde{C}_1$ as well as $C_k^H := \tilde{C}_k$. Similarly, for $j \in [k-1] \setminus \{1\}$, let $R_j^H \subseteq V(H)$ be the set of vertices that contains each vertex in $\tilde{R}_j$ and the vertices added to H for joining two vertices in $\tilde{R}_j$ that are adjacent in $\tilde{G}_k$. Define $R_1^H := \tilde{R}_1$ as well as $R_k^H := \tilde{R}_k$. By construction, the sets $C_1^H, \dots, C_k^H$ are pairwise disjoint and also the sets $R_1^H, \dots, R_k^H$ are pairwise disjoint. Furthermore, for each $i \in [k]$, the set C_i^H induces a path in H and, for each $j \in [k]$, the set R_j^H induces a path in H. Hence, H contains a $k \times k$ grid as a minor.

It is easy to verify that for all $i, j \in [k]$ the set $C_i^H \cap R_j^H$ induces a tree in H. Therefore, the graph H has exactly $(k-1)^2 + 1$ faces. Assume for a contradiction that H is not a minimal graph containing a $k \times k$ grid as a minor. Then, there is a graph $H' \subseteq H$ that contains a $k \times k$ grid as a minor and there is a vertex $v \in V(H)$ with $H' \subseteq H - v$ or there is an edge $e \in E(H)$ with $H' \subseteq H - e$. Since each vertex of H and each edge of H is contained in a row or a column of H that induces a path in H, each vertex of H is on the boundary of at least two faces of H and also each edge of H is on the boundary of at least two faces of H. Hence, H' has at most $(k-1)^2$ faces. As the number of faces does not increase when contracting H' to a $k \times k$ grid and modifying its embedding accordingly, the $k \times k$ grid would have at most $(k-1)^2$ faces, which is a contradiction. Therefore, H is a minimal graph containing a $k \times k$ grid as a minor.

By construction, H is a normal graph. So for each small face f of G_0 there is a small face f' of H that corresponds to f. Next, it is argued that H is a feasible subgraph of G_ϕ. First, consider a face f of G_0 that does not belong to a gadget or an extended gadget that is placed in G_0 in order to obtain G_ϕ. Then, for each subdivision vertex v_e that, in G_0, is on the boundary of f, the graph H either contains v_e or e. So, in H, there are at most 8 vertices embedded in the face f' corresponding to f and f' is feasible with respect to G_ϕ. If f is a variable face of G_0 or an unused connection face of G_0, then the face f' corresponding to f is feasible with respect to G_ϕ due to Claim 3.48c) and d). Consider a variable x and let f_1 be a connection face of x that is not an unused connection face. Claim 3.51c) says that there is a unique vertical line L in $\mathcal{R}$ such that $\hat{F}_L$ uses f_1. Let $C \in \mathcal{C}$ be the clause such that the last face of $\hat{F}_L$ is f_C. Furthermore, let $v_e \in V(G_0)$ be the subdivision vertex on the boundary of f_1 that is also on the boundary of a variable face f_0 of G_0. Note that f_0 is a bifurcation face of G_0 and, in the construction of G_ϕ, the subdivision vertex v_e has been replaced by its original edge e. If the assignment of T for x is good for C, then H relaxes along $\hat{F}_L$ and also along the face-sequence obtained from F_L by deleting the first face and the first subdivision vertex. Hence, the face of H corresponding to f_1 is feasible with respect to G_ϕ. Otherwise, i. e., if the assignment of T for x is not good for C, then the graph H does not use e and the grid-path of H that contains both neighbors of e uses the two new vertices embedded in the bifurcation face f_0. Furthermore, the graph H pulls along $\hat{F}_L$ and, hence, uses the edge inserted to split f_1 horizontally. Therefore the face of H that corresponds to f_1 is feasible with respect to G_ϕ. Moreover, for each face f of G_0 that is a wire face and neither a clause nor a connection face of an extended variable gadget, it is easy to see that the face of H that corresponds to f is feasible with respect to G_ϕ. Finally, consider a clause face f_C of G_0. As the assignment of T is a satisfying assignment for ϕ, there is a variable $x \in \mathcal{U}$ such that the assignment of T for x is good for C. Let L be the vertical line in $\mathcal{R}$ that joins x to C. Then, H is relaxing along $\hat{F}_L$ and the face of H that corresponds to f_C is feasible with respect to G_ϕ due to Claim 3.49b). □

Claim 3.55.
If there is a feasible subgraph $H \subseteq G_\phi$, then ϕ is satisfiable.

Proof. Let H be a feasible subgraph of G_ϕ and consider H as a plane graph with its induced embedding with respect to the embedding of the plane graph G_ϕ. This is not a restriction as G_ϕ and H are both uniquely embeddable due to Claim 3.53 and Remark 3.15, respectively. For each face f of G_ϕ, there is a unique face f' of H such that, for each vertex $v \in V(G_\phi)$ that is embedded in f, the vertex v is embedded in f'. In the following, we say that f' is the *face of H that contains f*. By construction, the graph G_ϕ contains a face f_{large} that is bounded by the cycle B_{large}. As $k \geq 5$, the cycle B_{large} contains at least 20 vertices and the face of H that contains f_{large} must be the large face f'_{large} of H. Since no vertex from $V(G_\phi) \setminus V(H)$ is embedded in f'_{large}, every vertex in B_{large} is on the boundary of f'_{large}.

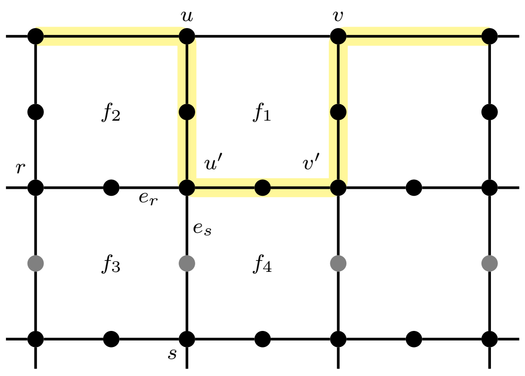


Figure 3.31: Notation in the proof of Claim 3.55. The boundary of the large face of G_0 is drawn on the top. For a contradiction, it is assumed that the boundary of H, which is highlighted, does not contain the edge $\{u, v\}$. At most two of the gray subdivision vertices have been replaced by their original edges when constructing G_ϕ from G_0.

For a contradiction, assume that there is an edge $e = \{u, v\}$ in B_{large} that is not contained in H. Then, u cannot be one of the vertices $(1,1)$, $(1,k)$, $(k,1)$, and (k,k) as otherwise $\deg_H(u) = 1$, but H does not contain such a vertex by Proposition 3.16. Let f_1 be the small face of G_0 with u and v on its boundary, and let u' and v' be the other two grid vertices on the boundary of f_1 such that u and u' are adjacent in $\tilde{G}_k$ and v and v' are adjacent in $\tilde{G}_k$. See Figure 3.31 for the following definitions. Let f_2 be the small face of G_0 with $f_2 \neq f_1$ and u' and u on its boundary. Denote by r the neighbor of u' in $\tilde{G}_k$ that is on the boundary of f_2 but is not u. Let f_3 be the small face of G_0 with $f_3 \neq f_2$ and u' and r on its boundary. Denote by s the neighbor of u' in $\tilde{G}_k$ that is also on the boundary of f_3 but is not r. Let f_4 be the small face of G_0 with u', v', and s on its boundary. When constructing G_ϕ from G_0, the faces f_1 and f_2 were not modified, and at most one of the faces in $\{f_3, f_4\}$ was modified to be an unused connection face due to the placement of the variable gadgets and Claim 3.52b). Moreover, the cycle bounding the face f'_{large} of H uses the grid vertex u, the subdivision vertex between u and u', the grid vertex u', the subdivision vertex between u' and v', the grid vertex v', the subdivision vertex between v' and v, as well as the grid vertex v as otherwise one of these subdivision vertices was embedded in the face f'_{large} of H. Then, u' is on the boundary of f'_{large} and Proposition 3.17b) implies that $\deg_H(u') \in \{2,3\}$. Let e_r be the edge of G_ϕ that connects u' to the subdivision vertex between u' and r and let e_s be the edge of G_ϕ that connects u' to the subdivision vertex between u' and s or, if the subdivision vertex between u' and s has been replaced by its original edge, let e_s be the edge that connects u' and s. So, H contains at most one of the edges e_r or e_s.

Case 1: H does not contain e_r. Let f' be the face of H where, in the drawing of G_ϕ, the edge e_r is embedded. Then, f' is a small face of H and f' contains f_2 and f_3. Therefore, each vertex that, in the drawing of G_ϕ, is embedded in f_2 or f_3 is in f'. In particular, these are 6 grid vertices and the three subdivision vertices on the boundary of f_2, as f_2 has not been modified when constructing G_ϕ from G_0 according to Claim 3.52b). Hence, there are at least 9 vertices in $V(G_\phi)$ that are embedded in f, which contradicts that H is a feasible subgraph of G_ϕ.

Case 2: H does not contain e_s. Let f' be the face of H where, in the drawing of G_ϕ, the edge e_s is embedded. Then, f' is a small face of H and contains f_3 and f_4. Therefore, each vertex that, in the drawing of G_ϕ, is embedded in f_3 or f_4 is in f'. In particular, these are 6 grid vertices and at least three subdivision vertices as at most one of the faces in $\{f_3, f_4\}$ is an unused connection face. Hence, there are at least 9 vertices in $V(G_\phi)$ that are embedded in f, which contradicts that H is a feasible subgraph of G_ϕ.

Consequently, H contains the edge $\{u, v\}$ and the face f'_{large} is bounded by B_{large}.

There are exactly $4(k-2)$ vertices v in B_{large} with $\deg_H(v) = 3$ and exactly 4 vertices v in B_{large} with $\deg_H(v) = 2$. Hence, none of the edges in B_{large} is contracted when contracting H to a $k \times k$ grid. Consider a partition of $V(H)$ into sets $M_{i,j}$ with $i, j \in [k]$ such that contracting the set $M_{i,j}$ to one vertex (i, j) results in the $k \times k$ grid. Due to Proposition 3.16, the vertex $(1, 1)$ is in one of the following sets: $M_{1,1}$, $M_{1,k}$, $M_{k,1}$, or $M_{k,k}$. Without loss of generality, we may assume that $(1, 1) \in M_{1,1}$ and also that $(i, j) \in M_{i,j}$ for each vertex (i, j) in B_{large}. Let $C^H_1, \dots, C^H_k$ and $R^H_1, \dots, R^H_k$ be the columns and rows of H that result from the choice of the sets $M_{i,j}$.

For a contradiction, assume that there is a grid vertex $v = (i, j) \in V(G_\phi)$ that is not in H. As each vertex in B_{large} is in H, it follows that $i \in [k-1] \setminus \{1\}$ and $j \in [k-1] \setminus \{1\}$. On the one hand, there are k vertex disjoint R^H_1,R^H_k-paths in H, since each column of H induces a R^H_1,R^H_k-path in H and these paths are pairwise vertex-disjoint. On the other hand, the set $\tilde{R}_j$ separates R^H_1 from R^H_k in G_ϕ due to Claim 3.52c) and, hence, the set $\tilde{R}_j \setminus \{(i, j)\}$ separates R^H_1 from R^H_k in H. So, there are k vertex disjoint R^H_1,R^H_k-paths in H and there is a set of $k-1$ vertices that separates R^H_1 from R^H_k in H, which contradicts Lemma 3.43. Consequently, each grid vertex is in H. Furthermore, it follows that each grid vertex $v = (i, j)$ is in one column $C^H_{i'}$ for some $i' \in [k]$. Since H is planar, the columns of H cannot cross and therefore $i = i'$, i. e., the grid vertex (i, j) is in the column C^H_i. Analogously, it follows that, for all $i, j \in [k]$, the grid vertex (i, j) is in the row R^H_j. Therefore, H is a normal subgraph of G_ϕ.

To define a truth assignment $\mathcal{U} \to \{\textsc{True}, \textsc{False}\}$, consider a variable $x \in \mathcal{U}$ and let $i \in [k]$ be the index such that the i^{th} column of G_0 is the decision column defined by the variable gadget corresponding to x. The grid-path $H[C^H_i]$ is called the *grid-path of the variable* x. According to Claim 3.48e), the grid-path of the variable x uses either the positive or the negative edge of x. Set $T(x) = \textsc{True}$ if and only if the grid-path of x uses the positive edge of x. For a contradiction, assume that there is a clause $C \in \mathcal{C}$ that is not satisfied by the assignment T, i. e., C evaluates to $\textsc{False}$ when plugging in the values for the variables according to T. Assume that C is a positive clause, the following is easy to adjust for a negative clause. Let x be an arbitrary variable used in C and denote by L_x a line in $\mathcal{R}$ that joins x to C. Then $T(x) = \textsc{False}$ and, hence, the grid-path of x uses the negative edge of x. Let $F_{L_x} = (f_0, v_{e_1}, f_1, \dots, v_{e_d}, f_d)$. Then, $f_d = f_C$ and f_0 and f_1 are the negative and the positive face of the decision gadget corresponding to x, respectively. Furthermore, the subdivision vertex v_{e_1} has been deleted when constructing G_ϕ and, between the neighbors of v_{e_1}, the grid-path of x uses only vertices embedded in f_0. By Claim 3.52d), the graph G_ϕ forces to pull along F_{L_x} and therefore H is pulling along F_{L_x}. As x was an arbitrary variable used in C and C contains exactly three literals by assumption, there are three distinct face-sequences F that each end in f_C and such that H pulls along F. Moreover, these three face-sequence are disjoint except for the last face due to Claim 3.51b). Then the face of H corresponding to f_C is not feasible with respect to G_ϕ by Claim 3.49a), which contradicts that H is a feasible subgraph of G_ϕ. Consequently, C is satisfied and T is a satisfying assignment for ϕ. □

This completes the proof of Theorem 3.31.

3.3.5 Proof of Approximability Results for the HEG Problem

Here, the positive results on the approximability of the Min HEG Problem are proved. First, Lemma 3.29 that claims that there is a 9-approximation for the Min HEG Problem is proved.

Algorithm 3.2: Computes an embedding that corresponds to a 9-approximation of the Min HEG Problem.

```
Input: an instance $(k, G, H)$ of the Min HEG Problem such that no edge in $E(G) \setminus E(H)$ joins
       two vertices in $H$.
Output: an embedding of $G$ that is an extension of the embedding of $H$ such that no vertex
        of $V(G) \setminus V(H)$ is embedded in the large face of $H$, or $\ell = \infty$ if $G$ does not have an
        embedding with the required properties.
1 Determine the large face $f_{\text{large}}$ of $H$ and compute a list of the vertices on the boundary of $f_{\text{large}}$;
2 Let $\hat{G}$ be the graph obtained from $G$ by adding a new vertex $v^*$ that is adjacent to all vertices on the
    boundary of $f_{\text{large}}$;
3 If $\hat{G}$ is not planar then Return $\ell = \infty$ ;
4 Compute an embedding $\hat{\mathcal{G}}$ of $\hat{G}$ in the plane;
5 Modify $\hat{\mathcal{G}}$ such that $\hat{\mathcal{G}}$ remains planar and no vertex of $V(G) \setminus V(H)$ is embedded in $f_{\text{large}}$;
6 Let $\mathcal{G}$ be the embedding of $G$ that is obtained from $\hat{\mathcal{G}}$ by removing $v^*$ and all edges incident to $v^*$;
7 Return the embedding $\mathcal{G}$;
```

Proof of Lemma 3.29. Consider an instance (k, G, H) of the Min HEG Problem, denote by ℓ its optimal value, and assume that $\ell < \infty$. Fix an embedding $\mathcal{G}_{\text{opt}}$ of G that corresponds to an optimal solution for the instance (k, G, H). Let $\mathcal{G}$ be an arbitrary embedding of G where no vertex of $V(G) \setminus V(H)$ is embedded in the large face of the induced embedding of H. Note that $\mathcal{G}$ is an extension of H as required by the Min HEG Problem, since H is uniquely embeddable due to Remark 3.15. The aim is to show that, in $\mathcal{G}$, each small face of the induced embedding of H contains at most 9ℓ vertices of G. Consider a small face f of H. Let U be the set of vertices of H that are on the boundary of f. Each component of $G - V(H)$ that, in $\mathcal{G}$, is embedded in f contains at least one vertex that has a neighbor in U because G is connected. There are at most 8 other small faces of H that contain a vertex in U on their boundary, namely the small faces surrounding f. Let F be the set of these small faces of H. Consider a component $\tilde{G}$ of $G - V(H)$ that, in $\mathcal{G}$, is embedded in f. In $\mathcal{G}_{\text{opt}}$, the component $\tilde{G}$ is embedded in f or in one of the faces in F. As, in $\mathcal{G}_{\text{opt}}$, there are at most 9ℓ vertices of G that are embedded in f or in one of the faces in F, it follows that, in $\mathcal{G}$, there are at most 9ℓ vertices of G embedded in f. Thus, $\mathcal{G}$ corresponds to a 9-approximation of the Min HEG Problem.

From now on, assume that there is no edge in $E(G) \setminus E(H)$ that joins two vertices of H. In the remainder of the proof, an embedding of G is called *feasible* if it is an extension of the embedding of the plane graph H such that no vertex of $V(G) \setminus V(H)$ is embedded in the large face of H. Algorithm 3.2 is applied to construct a feasible embedding of G. First, it is argued that the procedure described in Algorithm 3.2 can always be executed. Denote by $\hat{G}$ the graph constructed in Line 2 and let v^* be the vertex added to G when constructing $\hat{G}$. To see that the procedure is feasible, recall that the large face of a minimal graph containing a $k' \times k'$ grid with $k' \geq 5$ as a minor is unique. Thus, Line 1 is feasible. Other than that, it only remains to argue that Line 5 is feasible, if reached. So assume that $\hat{G}$ is planar and let $\hat{\mathcal{G}}$ be an embedding of $\hat{G}$ in the plane. If, in $\hat{\mathcal{G}}$, no vertex in $V(G) \setminus V(H)$ is embedded in the large face of H, then there is nothing to do in Line 5. Otherwise, note that the large face of H can be partitioned into several regions, such that each region is bounded by a cycle (v^*, x, y) in $\hat{G}$, where x and y are adjacent vertices on the boundary of the large face of H. Let R_{large} be the set of these regions. To argue that Line 5 is feasible, it suffices to consider each region $r \in R_{\text{large}}$ and move every vertex in $V(G) \setminus V(H)$ that is embedded in r to a small face of H. Let $r \in R_{\text{large}}$ and denote by x and y the two vertices other

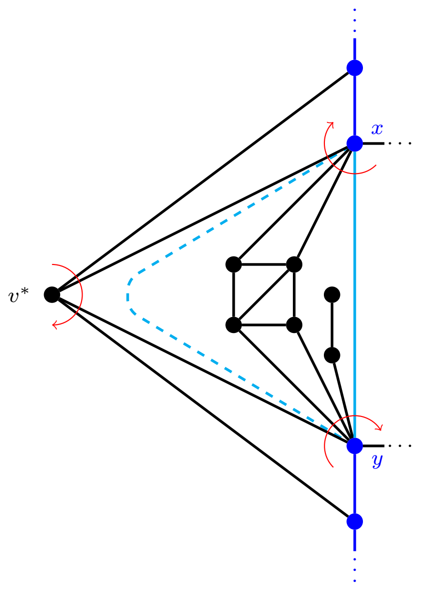
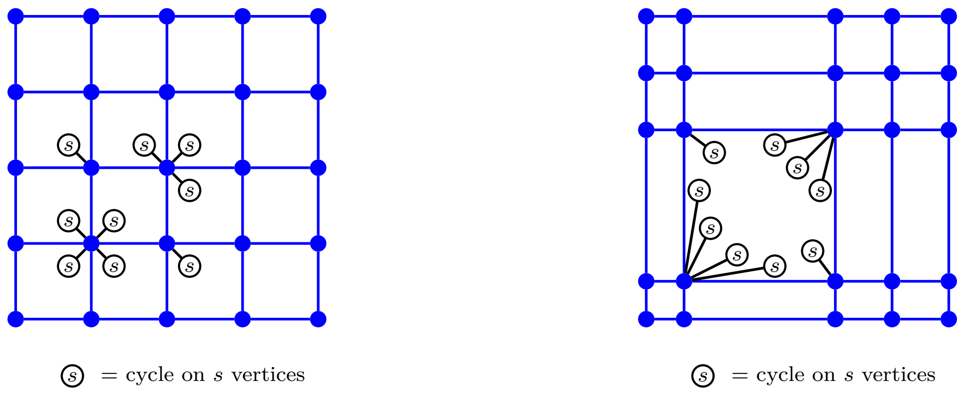


Figure 3.32: Adjusting the embedding in Line 5 of Algorithm 3.2. Blue vertices and edges belong to the subgraph H. The dotted line indicates a possibility to draw the light blue edge such that no vertex of $G - V(H)$ is embedded inside the cycle (v^*, y, x). The red arrows visualize the circular ordering of the vertices v^*, x, and y.

than v^* on the boundary of r such that the edge $\{v^*, y\}$ is the edge after $\{v^*, x\}$ in the circular ordering of v^*. In the embedding $\hat{\mathcal{G}}$, the edge $\{x, y\}$ can also be drawn close to the edges $\{v^*, x\}$ and $\{v^*, y\}$ so that all vertices of $V(G) \setminus V(H)$ are embedded on the same side of the cycle (v^*, x, y) and the embedding remains planar, see Figure 3.32. Applying the same procedure to every region in R_{large} shows that Line 5 is feasible. Furthermore, when an embedding $\mathcal{G}$ of G is returned, then it is feasible. Indeed, embeddings are stored by the corresponding rotation systems, i. e., the algorithm only keeps track of the combinatorial structure of the embedding. Hence, Remark 3.15 implies that $\mathcal{G}$ is an extension of the embedding of H and due to Line 5, in $\mathcal{G}$, no vertex of $V(G) \setminus V(H)$ is embedded in the large face of H.

To show that Algorithm 3.2 is correct, i. e., that a feasible embedding of G is computed if and only if G has a feasible embedding, define $\hat{G}$ and v^* as above. Assume that G admits a feasible embedding $\mathcal{G}$. Then, in $\mathcal{G}$, there is no vertex from $V(G) \setminus V(H)$ embedded in the large face of H and there is no edge of $E(G) \setminus E(H)$ embedded in the large face of H, as no edge in $E(G) \setminus E(H)$ joins two vertices in H. Hence, the large face of H is also a face of $\mathcal{G}$, the graph $\hat{G}$ is planar, and an embedding of G is computed. As argued above, the computed embedding of G is feasible. Now, assume that the algorithm returns an embedding $\mathcal{G}$ of G. Again, $\mathcal{G}$ is feasible and this implies that G admits a feasible embedding.

It remains to analyze the running time of Algorithm 3.2. Let n be the number of vertices of G and assume that $V(G) = [n]$ as always for implementations. Due to Lemma 2.26, which can be applied as $\Delta(H) \leq 4$ by Proposition 3.16, a list of the faces of H and a list of the vertices on the boundary of each face f of H can be computed together in $\mathcal{O}(\|H\|)$ time. Due to Proposition 3.17, the large face of H is the unique face of H that contains $4(k-2) \geq 12$ vertices v with $\deg_H(v) = 3$. So, to find the large face of H, the algorithm determines the degree of each vertex in H as well as, for each face f of H, the number of vertices v with $\deg_H(v) = 3$ on the boundary of f. This takes $\mathcal{O}(\|H\|)$ time as all lists of vertices on the boundaries of the faces of H were computed in $\mathcal{O}(\|H\|)$ time and, hence, can also be traversed in $\mathcal{O}(\|H\|)$ time. Denote by L_{large} the list of vertices on the boundary of the large face of H. Then, the graph $\hat{G}$ in



a) An optimal solution for the Min HEG Problem, i. e., each small face of H contains at most ℓ_{opt} vertices.

b) An embedding of G where one small face of H contains roughly $9\ell_{\text{opt}}$ vertices.

Figure 3.33: A graph G and different embeddings of G. The blue subgraph H is a minimal graph containing a 5×5 grid as a minor. Denote by ℓ_{opt} the optimal value for the instance (k, G, H) with $k = 5$ of the Min HEG Problem.

Line 2 can be computed from G by adding a new vertex $v^* = n+1$ whose adjacency list is L_{large} and adding v^* to the adjacency list of each vertex in the list L_{large}, which takes time proportional to the length of L_{large}, which is $\mathcal{O}(|V(H)|) = \mathcal{O}(\|H\|)$ time. Furthermore, $\|\hat{G}\| \leq \|G\| + 1 + |V(H)| \leq 3\|G\|$. Hence, it follows from Theorem 2.28 that Line 3 and Line 4 take $\mathcal{O}(\|G\|)$ time. To implement Line 5, the algorithm does the following for each pair (x, y) with the property that, in $\hat{\mathcal{G}}$, the edge $\{v^*, y\}$ is the edge after $\{v^*, x\}$ in the circular ordering of x in the embedding $\hat{\mathcal{G}}$. To adjust the rotation systems representing $\hat{\mathcal{G}}$ in a way that corresponds to moving the edge $\{x, y\}$ in the embedding $\hat{\mathcal{G}}$ as described above, in the rotation system of x, the algorithm moves the entry y to be the entry before v^* and, in the rotation system of y, the algorithm moves the entry x to be the entry after v^*. Hence, the processing time for one pair (x, y) is proportional to $2 + \deg_G(x) + \deg_G(y)$. As only adjacency lists of vertices in L_{large}, i. e., adjacency lists of vertices on the boundary of the large face of H, are rearranged, and for each such vertex, the adjacency list is rearranged exactly two times, Line 5 takes time proportional to

$$\sum_{x \text{ in } L_{\text{large}}} (1 + \deg_G(x)) \;\leq\; |V(G)| + 2|E(G)| \;\leq\; 2\|G\|.$$

Consequently, the total running time is $\mathcal{O}(\|H\| + \|G\|) = \mathcal{O}(\|G\|) = \mathcal{O}(n)$ due to Corollary 2.9. $\square$

The approximation ratio of the algorithm in Lemma 3.29, i. e., the approximation ratio of Algorithm 3.2, is tight as the following example shows. Let H be the 5×5 grid with an embedding in the plane and fix an integer $s \geq 3$. Let G be the graph obtained from H by attaching nine cycles, each on s vertices, in the following way: four cycles at the vertex $(2, 2)$, three cycles at the vertex $(3, 3)$, one cycle at the vertex $(2, 3)$, and one cycle at the vertex $(3, 2)$. Each of the nine cycles is attached by joining one of its vertices to the corresponding vertex of the grid with a new edge. Clearly, G is planar and H is a minimal graph containing a 5×5 grid as a minor. Denote by ℓ_{opt} the optimal solution for the instance (k, G, H) with $k = 5$ of the Min HEG Problem. Figure 3.33a) shows that there is an embedding of G such that each small face of H contains at most $s + 4$ vertices of G and no vertex in $V(G) \setminus V(H)$ is embedded in the large face of H, i. e., $\ell_{\text{opt}} \leq s + 4$. Moreover, there is an embedding of G, where all vertices that are not in H are embedded in the same small face f of H, namely the face f that is bounded by the vertices $(2,2)$, $(3,2)$, $(3,3)$, and $(2,3)$, see Figure 3.33b). Then, there are $\ell = 9s+4$ vertices of G embedded in f. Thus, any value $\alpha \in \mathbb{R}$ for which Lemma 3.29 is true when replacing 9-approximation by α-approximation must satisfy $\ell \leq \alpha \cdot \ell_{\text{opt}}$. Hence,

$$\alpha \geq \frac{\ell}{\ell_{\text{opt}}} \geq \frac{9s+4}{s+4} \quad \xrightarrow{s\to\infty} \quad 9,$$

and the approximation ratio in Lemma 3.29 is tight with respect to the construction used there.

Next, the proof for Theorem 3.30, which states that there is a 5-approximation for the Simplified Min HEG Problem, is presented. The following lemma about computing disjoint paths in planar graphs will be useful as it helps to analyze the structure of the minimal graph containing a $k\times k$ grid as a minor.

Lemma 3.56 (Ripphausen-Lipa, Wagner, Weihe [RLWW97]).
Let $G=(V,E)$ be a planar graph on n vertices and fix two distinct vertices $x,y \in V$. There is an algorithm that determines the maximum number of pairwise internally vertex-disjoint x,y-paths in G and returns such a collection of x,y-paths in $\mathcal{O}(n)$ time.

In [RLWW97], it is also mentioned that the case that is interesting for the proof of Theorem 3.30 is particularly easy. Consider a planar graph $G=(V,E)$ on n vertices and fix two distinct vertices $x,y \in V$. When Lemma 3.56 is applied here, there is an embedding of G that contains a face f such that the vertices x and y are on the boundary of f. In [RLWW97], such a graph G is called (x,y)*-planar*. To find a maximum collection of disjoint x,y-paths in G, one can construct x,y-paths one by one in the following way until the remaining graph does not contain an x,y-path anymore. The algorithm starts in x and traverses the boundary of the face f until y is reached, similarly as when determining the boundaries of the faces of a graph in Lemma 2.26. Therefore, an x,y-walk is obtained and, if necessary, this x,y-walk can be shortened to an x,y-path. Before constructing the next x,y-path, the algorithm removes all vertices discovered in the x,y-walk, except for x and y. Since every edge of G and every vertex in $V \setminus \{x,y\}$ is used in at most one step of the construction, the algorithm runs in $\mathcal{O}(n)$ time by Corollary 2.9. Note that an embedding of G can be computed in $\mathcal{O}(n)$ time by Theorem 2.28 and when temporarily adding the edge $\{x,y\}$ to G, if it is not yet in G, the resulting embedding of G will contain a face f such that x and y are on the boundary of f.

Proof of Theorem 3.30. Let (k,G,H) be an instance of the Simplified Min HEG Problem. Denote by ℓ_{opt} its optimal value, and let $\mathcal{G}_{\text{opt}}$ be a corresponding embedding of G. The idea to construct an embedding $\mathcal{G}$ of G, that can be computed in polynomial time, is to start with the embedding of H and add the components of $G-V(H)$ one by one. When doing so, consider a force that pulls the components of $G-V(H)$ in the direction of the vertices corresponding to the vertex $(1,1)$ of the $k\times k$ grid to which H can be contracted. More precisely, consider a partition of $V(H)$ into non-empty sets $M_{i,j}$ with $i,j \in [k]$ as in Proposition 3.24. For $i,j \in [k-1]$, the properties stated there imply that the subgraph of H induced by $M_{i,j} \cup M_{i+1,j} \cup M_{i+1,j+1} \cup M_{i,j+1}$ contains a unique cycle that bounds a small face of H, say $f_{i,j}$. Note that each small face of H is a face $f_{i,j}$ for some $i,j \in [k]$. Consider an arbitrary component $\tilde{G}$ of $G-V(H)$. As G is connected, there is a vertex u in H that has a neighbor w in $V(\tilde{G})$. Observe that u and w are both unique, as otherwise the edge $\{u,w\}$ could be removed from G without destroying the connectivity of G and (k,G,H) would not be an instance of the Simplified Min HEG Problem. Furthermore, there are unique indices $i,j \in [k]$ with $u \in M_{i,j}$. As each vertex in $M_{i,j}$ can only be on the boundary of the faces $f_{i-1,j-1}$, $f_{i,j-1}$, $f_{i,j}$, and $f_{i-1,j}$, there are at most four possibilities to embed $\tilde{G}$ and the algorithm tries them in the following order

$$1.\ f_{i-1,j-1} \qquad 2.\ f_{i,j-1} \qquad 3.\ f_{i-1,j} \qquad 4.\ f_{i,j}. \tag{3.16}$$

Algorithm 3.3: 5-approximation for the Simplified Min HEG Problem including a corresponding embedding.

```
Input: an instance $(k, G, H)$ of the Simplified Min HEG Problem.
Output: an integer $\ell$ with $\ell \leq 5\ell_{\text{opt}}$, where $\ell_{\text{opt}}$ denotes the optimal solution for $(k, G, H)$ and a
        corresponding embedding of $G$.
1 Label the small faces of $H$ with $f_{i,j}$ for $i, j \in [k-1]$ as done in the construction;
2 For all $i, j \in [k-1]$ let $n_{i,j}$ be the number of vertices on the boundary of $f_{i,j}$;
3 Let $\mathcal{G}$ be the embedding of the plane graph $H$;
4 ForEach component $\tilde{G}$ of $G - V(H)$ do
5     Add $\tilde{G}$ to $\mathcal{G}$ according to (3.16) and update the corresponding value $n_{i,j}$;
6 EndFch
7 $\ell \leftarrow \max\{n_{i,j} \colon i, j \in [k-1]\}$;
8 Return $\ell$ and the embedding $\mathcal{G}$;
```

Observe that not all of these options must be feasible, as the vertex u is not necessarily on the boundary of all these faces and not all of these faces exist for all values of $i, j \in [k]$. Nevertheless, as G is planar, there is at least one small face f of H where $\tilde{G}$ can be embedded, i. e., the vertex u is on the boundary of f, and this face f is one of the faces listed in (3.16). Furthermore, for each face f in the list in (3.16), that exists and is such that u is on the boundary of f, the component $\tilde{G}$ can be embedded in f. This is independent of previous choices to embed other components of $G - V(H)$ as (k, G, H) is an instance of the Simplified Min HEG Problem. Let $\mathcal{G}$ be the embedding of G that is obtained by applying these rules to embed all components of $G - V(H)$. Algorithm 3.3 summarizes this construction, where $n_{i,j}$ for $i, j \in [k-1]$ denotes the number of vertices of $V(G)$ that are embedded in the face $f_{i,j}$.

Denote by ℓ the integer computed by Algorithm 3.3 when applied to (k, G, H). Next, it is argued that $\ell \leq 5\ell_{\text{opt}}$, which means that Algorithm 3.3 is a 5-approximation for the Simplified Min HEG Problem. Consider a small face $f_{i,j}$ of H for arbitrary $i, j \in [k-2] \setminus \{1\}$. The following argument is easy to adjust for all other $i, j \in [k-1]$. In the proof of Lemma 3.29, it was argued that, in an arbitrary embedding of G that is an extension of the embedding of H, only components of $G - V(H)$ that, in $\mathcal{G}_{\text{opt}}$, are embedded in the faces

$$f_{i-1,j-1},\ f_{i,j-1},\ f_{i+1,j-1},\ f_{i-1,j},\ f_{i,j},\ f_{i+1,j},\ f_{i-1,j+1},\ f_{i,j+1},\ \text{and}\ f_{i+1,j+1}$$

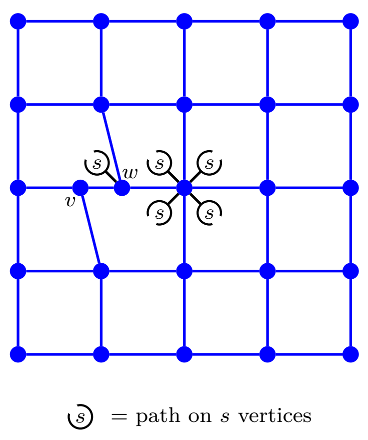
of H can be embedded in $f_{i,j}$. However, in the embedding $\mathcal{G}$, vertices that can be embedded in $f_{i-1,j-1}$, $f_{i,j-1}$, $f_{i+1,j-1}$, or $f_{i-1,j}$ will not be embedded in $f_{i,j}$. Hence, in $\mathcal{G}$, at most all components of $G - V(H)$ that, in $\mathcal{G}_{\text{opt}}$, are embedded in the faces $f_{i,j}$, $f_{i+1,j}$, $f_{i-1,j+1}$, $f_{i,j+1}$, or $f_{i+1,j+1}$ of H are embedded in $f_{i,j}$. Consequently, in $\mathcal{G}$, at most $5\ell_{\text{opt}}$ vertices of G are embedded in the face $f_{i,j}$ of H and Algorithm 3.3 is a 5-approximation for the Simplified Min HEG Problem.

Next, it is argued that the procedure described in Algorithm 3.3 can be implemented to run in $\mathcal{O}(n\sqrt{n})$ time, where n denotes the number of vertices of G. Due to Lemma 2.26 and Lemma 2.27, which can be applied because $\Delta(H) \leq 4$ by Proposition 3.16, for each face f of the plane graph H, a list L_f of the vertices on the boundary of f, and, for each vertex $u \in V(H)$, a list L_u of faces whose boundary contains u can be computed in $\mathcal{O}(\|H\|)$ time. So, to implement Lines 1-3, it is only missing to label the faces as in the above construction. To do so, first, columns and rows of H are defined. For all $i, j \in [k]$ such that (i, j) is a vertex of the $k \times k$ grid of degree 3, i. e., for all $i, j \in [k]$ with either $i \in \{1, k\}$ or $j \in \{1, k\}$, let $v_{i,j}$ be a vertex in $M_{i,j}$ with $\deg_H(v_{i,j}) = 3$. Such a vertex exists and is unique due to Proposition 3.16. For $(i, j) \in \{(1,1), (1,k), (k,1), (k,k)\}$, let $v_{i,j}$ be an arbitrary vertex in $M_{i,j}$. In the following, the vertices $v_{i,j}$ that have been defined for certain values of $i, j \in [k]$ are called the *special vertices of the large face of* H as each of them lies on the boundary of the large face of H. For $i \in [k]$, let the i^{th} column C_i be the vertex set of the unique $v_{i,1}$,$v_{i,k}$-path in H that uses only vertices in $M_{i,j}$ for $j \in [k]$. For $j \in [k]$, let the j^{th} row R_j be the vertex set of the unique $v_{1,j}$,$v_{k,j}$-path in H that uses only vertices in $M_{i,j}$ for $i \in [k]$. Such paths exist and are unique due to Proposition 3.24. Furthermore, by construction, the sets $C_1, \dots, C_k$ are pairwise disjoint and so are the sets $R_1, \dots, R_k$. Note that these definitions depend on the choice of the sets $M_{i,j}$. If H was a $k \times k$ grid, then the sets $M_{i,j}$ can be chosen in a way such that, for each $i \in [k]$, the i^{th} column of H as defined for grids is exactly the set C_i defined here and similarly, for each $j \in [k]$, the j^{th} row of H as defined for grids is exactly the set R_j as defined here.

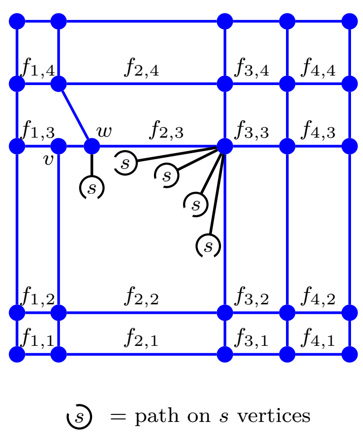
Now, if the algorithm computed a feasible choice for the columns $C_1, \dots, C_k$ of H and the rows $R_1, \dots, R_k$ of H, then it can label the faces as desired by using the following. For $i, j \in [k-1]$, the face $f_{i,j}$ is the unique face for which the set of vertices on its boundary intersects each of the sets C_i, C_{i+1}, R_j, and R_{j+1} and no other columns or rows of H. Observe that the set of vertices on the boundary of the large face of H intersects every column and every row of H. To label the faces, recall that $V(G) = [n]$ as always and $V(H) \subseteq [n]$. The algorithm initializes two integer arrays A_C and A_R of length n with zeros. Then, for every $i \in [k]$, it sets the entries of A_C that correspond to the vertices in C_i to i and similarly, for every $j \in [k]$, it sets the entries of A_R that correspond to the vertices in R_j to j, which takes $\mathcal{O}(n)$ time. Then, for each face f of H, it traverses the list L_f to determine the indices of the columns and rows that contain a vertex in L_f, i. e., that intersect the set of vertices on the boundary of f. If exactly two indices of columns, say i_1 and i_2, and exactly two indices of rows, say j_1 and j_2, are determined, then f is the small face $f_{i,j}$ of H with $i = \min\{i_1, i_2\}$ and $j = \min\{j_1, j_2\}$. Otherwise, f is the large face of H. Hence, when a face f is processed and a third index of a column or a third index of a row is encountered, the algorithm stops processing f as f is the large face. Therefore, processing one face f takes time proportional to the number of vertices on the boundary of f. As mentioned above, the lists L_f for all faces f of H can be computed in $\mathcal{O}(\|H\|)$ time and, hence, they can also be traversed in $\mathcal{O}(\|H\|)$ time. So, if one feasible choice for the columns and rows of H is known, the faces can be labeled in $\mathcal{O}(\|H\|)$ time.

Next, it is discussed how the algorithm computes a feasible choice of the columns and rows of H, as this information cannot be read off the embedding of H directly. The algorithm tries all possibilities to choose the special vertices of the large face of H. For each such choice, the algorithm checks if k vertex-disjoint paths can be found, such that each path is a $v_{i,1}$,$v_{i,k}$-path for some $i \in [k]$, i. e., whether there are columns $C_1, \dots, C_k$ for the choice of special vertices of the large face of H. To do so, the algorithm adds a new vertex $x = n + 1$ to H that is incident to $v_{i,1}$ for all $i \in [k]$ as well as a new vertex $y = n + 2$ that is incident to $v_{i,k}$ for all $i \in [k]$. Observe that H' is planar as x and y can both be embedded in the large face of H. If, in H, there are k vertex-disjoint paths, such that each path is a $v_{i,1}$,$v_{i,k}$-path for some $i \in [k]$, then there are k internally vertex-disjoint x,y-paths in H'. On the other hand, if there are k internally vertex-disjoint x,y-paths in H', then each vertex $v_{i,1}$ with $i \in [k]$ is contained in one of these paths and each vertex $v_{i,k}$ is contained in one of these paths as $\deg_{H'}(x) = \deg_{H'}(y) = k$. For each $i \in [k]$, one of these paths contains $v_{i,1}$ and $v_{i,k}$, as otherwise two paths would cross but this is not possible as the paths are internally vertex-disjoint and H' is planar. Furthermore, $V(H') \subseteq [n+2]$ and, to satisfy $V(H') = [n+2]$, all vertices in $V(G) \setminus V(H)$ can be added to H' as isolated vertices. So to decide whether there are k vertex-disjoint paths in H, such that each path is a $v_{i,1}$,$v_{i,k}$-path for some $i \in [k]$ the algorithm can construct the graph H' and apply the algorithm contained in Lemma 3.56 to H' with vertices x and y, which takes $\mathcal{O}(n)$ time. Similarly, the algorithm checks if H contains k vertex-disjoint paths, such that each path is a $v_{1,j}$,$v_{k,j}$-path for some $j \in [k]$, i. e., whether there are rows $R_1, \ldots, R_k$ for the choice of special vertices of the large face of H. If the columns and the rows exist, the corresponding partition of $V(H)$ into the sets $M_{i,j}$ for $i, j \in [k]$ can be defined, such that each special vertex $v_{i,j}$ of the large face of H is in $M_{i,j}$, and such that, for all $i, j \in [k]$, the set $M_{i,j}$ contains every vertex in $C_i \cap R_j$. Thus, once the algorithm chose the special vertices of the large face of H, it takes $\mathcal{O}(n)$ time to check whether this choice is feasible. So how many possibilities are there to choose the special vertices of the large face of H? All special vertices of the large face of H have degree 3, except for $v_{1,1}$, $v_{1,k}$, $v_{k,1}$, and $v_{k,k}$, and there are exactly $4(k-2)$ vertices of degree 3 on the boundary of the large face of H due to Proposition 3.17b). Hence, once the vertex $v_{1,2}$ is chosen, there are only two ways to choose the remaining special vertices that have degree 3 in H, due to their order on the boundary of the large face, and it suffices to consider one of them due to symmetry. To choose $v_{1,2}$, there are $4(k-2)$ vertices of degree 3 on the boundary of the large face of H and again due to symmetry it suffices to consider $(k-2)$ consecutive ones, where consecutive refers only to the vertices of degree 3. Moreover, when walking along the boundary of the large face of H from $v_{1,2}$ to $v_{2,1}$ such that $v_{1,3}$ is not traversed, then each traversed vertex has degree 2 and it does not matter which one is chosen as $v_{1,1}$ and similarly for $v_{1,k}$, $v_{k,1}$, and $v_{k,k}$. As $k^2 \leq |V(H)| \leq n$, there are at most $\sqrt{n}$ choices for the special vertices of the large face of H that need to be considered. So, all in all, a feasible choice for the columns and rows of H can be determined in $\mathcal{O}(n\sqrt{n})$ time.

To implement the loop in Lines 4-6, the algorithm first computes a list of the vertices in H and sets up a binary array A_H of length n, in which the entries corresponding to vertices of H are set to one, which takes $\mathcal{O}(n + \|H\|)$ time when Lemma 2.25 is used to compute a list of the vertices in H. For each vertex $u \in V(H)$ the algorithm determines the small face f of H in which all components $\tilde{G}$ of $G - V(H)$ such that, in G, there is an edge joining u to a vertex in $V(\tilde{G})$, are embedded, i. e., it chooses one face f_u from the list L_u according to (3.16). As there are at most four faces in the list L_u, this takes constant time for one node u and, hence, in total $\mathcal{O}(|V(H)|)$ time. Let $\mathcal{G}$ be the embedding of the plane graph H as initialized in Line 3. For each small face f of H, the algorithm traverses the list L_f and, for each vertex u in L_f with $f_u = f$, it does the following. The algorithm cyclically shifts the rotation system of u in the representation of $\mathcal{G}$ such that u' is the last vertex in the rotation system of u, where u' is the vertex before u in L_f or, if u is the first vertex in L_f, then u' is the last vertex of L_f. Consequently, when embedding a new edge $\{u, w\}$ in the face f_u, the adjustment of the rotation system of u reduces to appending w at the end of the rotation system of u. For each vertex u, the cyclical shift of its rotation system takes time proportional to $\deg_H(u)$, which is at most 4 due to Proposition 3.16. Hence, all shifts together can be done in time proportional to the sum of the lengths of the lists L_f, which is $\mathcal{O}(\|H\|)$ time. Then, the algorithm computes the graph $\hat{G} := G - E(H)$ by going through the adjacency lists of G and whenever an adjacency list of a vertex $u \in V(H)$ is traversed all entries $u' \in V(H)$ in the adjacency list of u are deleted, which takes $\mathcal{O}(\|G\|)$ time as the array A_H can be used to determine in constant time whether a vertex is in H. Observe that each component of $\hat{G}$ contains exactly one vertex of H as G is connected and $G - e$ is not connected for every $e \in E(G) \setminus E(H)$. Moreover, $V(\hat{G}) = V(G) = [n]$ and, hence, an embedding $\hat{\mathcal{G}}$ of $\hat{G}$ can be computed in $\mathcal{O}(n)$ time according to Theorem 2.28. Next, the embedding $\hat{\mathcal{G}}$ is added to the embedding $\mathcal{G}$ to obtain an embedding of G, or more precisely, $\hat{\mathcal{G}}$ contains a drawing of each component of $G - V(H)$ with its unique edge joining it to H and these drawings are now embedded in the faces of the plane graph H. To do so, for each vertex $u \in V(H)$, the algorithm appends the rotation system of u, that refers to the embedding $\hat{\mathcal{G}}$, at the end of the rotation system of u, that refers to the embedding $\mathcal{G}$. For each vertex $u \in V(G) \setminus V(H)$ the algorithm keeps the rotation system of u that refers to the embedding of $\hat{\mathcal{G}}$. This finishes the implementation of the loop in Lines 4-6, which takes $\mathcal{O}(\|H\| + \|G\|)$ time in total.



a) An optimal solution for the Simplified Min HEG Problem, i. e., each small face of H contains at most ℓ_{opt} vertices.



b) An embedding of G that is obtained with Algorithm 3.3.

Figure 3.34: A graph G and different embeddings of G. The blue subgraph H is a minimal graph containing a 5×5 grid as a minor. Denote by ℓ_{opt} the optimal value of the instance (k, G, H) with $k = 5$ of the Simplified Min HEG Problem.

To finish, Line 7 can be executed in time proportional to $k^2 \leq n$. So, the total running time of Algorithm 3.3 is

$$\mathcal{O}(\|H\| + \|G\| + n\sqrt{n}) \;=\; \mathcal{O}(\|G\| + n\sqrt{n}) \;=\; \mathcal{O}(n\sqrt{n})$$

by Corollary 2.9. □

The approximation ratio of the algorithm in Theorem 3.30, i. e., the approximation ratio of Algorithm 3.3, is tight as the following example shows. Fix an integer $s \geq 3$ and let H be the plane graph that is obtained from an embedding of the 5×5 grid when splitting the vertex $(2,3)$ into two vertices v and w of degree 3 such that H has one face bounded by the cycle $((1,2),(2,2),v,(1,3))$, see Figure 3.34. Let G be the graph obtained from H by attaching four paths, each on s vertices, to the vertex $(3,3)$ and one path on s vertices to the vertex w, where each path is attached by joining one of its vertices to the corresponding vertex of H. Clearly, H is a minimal graph containing a 5×5 grid as a minor and G is planar. So (k, G, H) is an instance of the Simplified Min HEG Problem for $k = 5$ and we denote by ℓ_{opt} its optimal value. The embedding of G in Figure 3.34a) shows that there is an embedding of G such that each small face of H contains at most $s + 5$ vertices of G and no vertex in $V(G) \setminus V(H)$ is embedded in the large face of H, i. e., $\ell_{\text{opt}} \leq s + 5$. Moreover, when applying Algorithm 3.3 to extend the embedding of H to an embedding of G and labeling the faces as in Figure 3.34b), all vertices that are not in H are embedded in the face $f_{2,2}$. So, there are $\ell = 5s + 5$ vertices of G embedded in $f_{2,2}$. Thus, the approximation ratio α of Algorithm 3.3 must satisfy $\ell \leq \alpha \cdot \ell_{\text{opt}}$ and therefore,

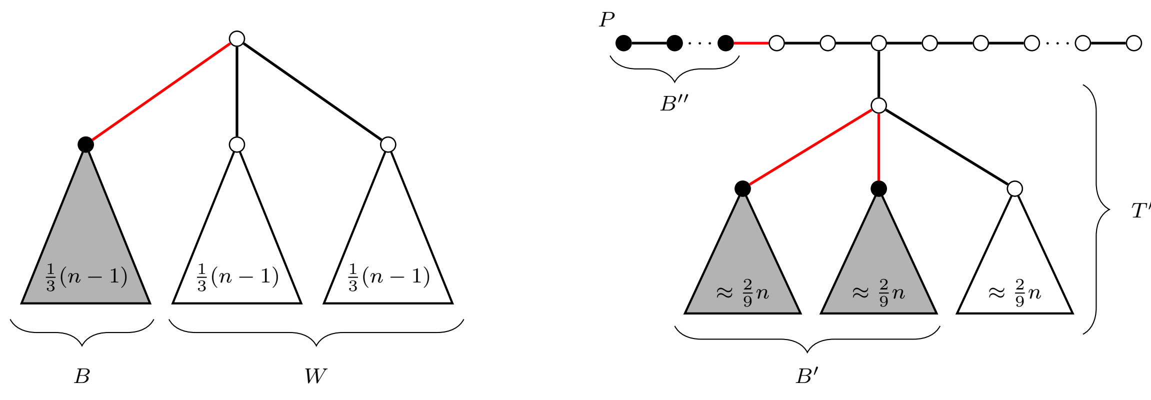
$$\alpha \;\geq\; \frac{\ell}{\ell_{\text{opt}}} \;\geq\; \frac{5s+5}{s+5} \quad \xrightarrow{s\to\infty} \quad 5.$$

Chapter 4

Approximate Cuts in Tree-Like Graphs

This chapter studies *approximate cuts*, which relax the size constraints compared to exact cuts. More precisely, consider a graph G on n vertices and an integer $m \in [n]$. We would like to find an m-cut (B, W) in G that cuts few edges, preferably only constantly many. This is not always possible, for example, consider a star on n vertices. Then every m-cut (B, W) cuts at least $\min\{m, n-m\}$ edges as one of the sets of the cut always consists of leaves only. Also, for $m = \lfloor \frac{1}{2}n \rfloor$, there is no m-cut of constant width in a perfect ternary tree on n vertices due to Theorem 2.4. Therefore, we relax the size constraints on the set B by requiring only $\frac{1}{2}m < |B| \leq m$ instead of $|B| = m$. In the example of a perfect ternary tree on $n \geq 13$ vertices, i. e., of height at least two, the cut (B, W) of width one, where the set B consists of the $\frac{1}{3}(n-1)$ vertices in the subtree rooted at a child of the root, is feasible now, see Figure 4.1a). In the case of the star on n vertices, the relaxation on the size constraints does not help much as each cut (B, W) with $\frac{1}{2}m < |B| \leq m$ cuts at least $\min\left\{\frac{1}{2}m, n-m\right\}$ edges. Therefore, this chapter focuses on bounded-degree graphs and shows that bounded-degree trees and bounded-degree tree-like graphs allow an approximate cut of constant width. The constant $\frac{1}{2}$ in the above example is the easiest case. Recall that, for $n \in \mathbb{N}$ and $m \in [n]$, a cut (B, W) in a graph G on n vertices is called a *simple approximate m-cut* if (B, W) satisfies $\frac{1}{2}m < |B| \leq m$ and, for a constant $c \in [0, 1)$, the cut (B, W) is called a *c-approximate m-cut* if (B, W) satisfies $cm \leq |B| \leq m$. Moreover, in this chapter, for $n \in \mathbb{N}$, $m \in [n]$, and for a constant $c \in [0, 1)$, a *strict c-approximate m-cut* is a cut (B, W) in a graph G on n vertices with the property that $cm < |B| \leq m$. The concept of strict c-approximate m-cuts will only be used in this chapter. Note that a simple approximate m-cut is the same as a strict $\frac{1}{2}$-approximate m-cut and every c-approximate m-cut with $c > \frac{1}{2}$ is a simple approximate m-cut but not vice versa. The aim of this chapter is to show that bounded-degree trees and bounded-degree tree-like graphs allow c-approximate m-cuts of small width for every feasible choice of the parameter m as well as that such a c-approximate m-cut can be computed in linear time.

Despite approximate cuts being interesting by themselves, our motivation to devote this chapter to approximate cuts is that they will be useful tools to construct exact cuts, in particular in Chapter 5. There, they are used to split up big pieces of the considered graph such that each of the resulting pieces will fit into one set of the final cut. For example, in Chapter 5, it is shown that every bounded-degree



a) A simple approximate $\lfloor \frac{1}{2}n \rfloor$-cut in a perfect ternary tree on n vertices.

b) A bisection in a tree T on n vertices with $\operatorname{diam}(T) \approx \frac{1}{3}n$, where T' is partitioned with a simple approximate cut.

Figure 4.1: Simple approximate cuts.

tree T on n vertices with $\operatorname{diam}(T) \geq \frac{1}{4}n$ allows a bisection of width at most $6\Delta(T)$, see also Theorem 1.1, which is more general. To demonstrate the use of a simple approximate cut to construct a bisection with these properties, fix an $n \in \mathbb{N}$ and consider the tree T that consists of a path P on $\lceil \frac{1}{3}n \rceil$ vertices and a perfect ternary tree T' on $\lfloor \frac{2}{3}n \rfloor$ vertices whose root is adjacent to one of the vertices in the middle of the path P, see Figure 4.1b). Then any bisection in T has to partition the vertex set of T' into two non-empty sets, as $V(T')$ is too large to fit completely into one set of the bisection. Theorem 2.4 says that at least $\Omega(\log n)$ edges are cut by every bisection in T'. However, this is not required: The aim is to find a bisection in T of width at most $6\Delta(T)$ and therefore a simple approximate m-cut with $m := \lceil \frac{1}{2}n \rceil$ suffices to partition the vertex set of T' into two sets (B', W') with $\frac{1}{4}n < |B'| \leq \lceil \frac{1}{2}n \rceil$. Then, it is easy to find a set $B'' \subseteq V \setminus V(T')$ such that the cut (B, W) with $B := B' \mathbin{\dot\cup} B''$ and $W := V \setminus B$ is a bisection in T that cuts few edges.

This chapter is organized as follows: Section 4.1 focuses on trees and shows that every bounded-degree tree T on n vertices allows a simple approximate m-cut of width at most $\Delta(T)$ for every $m \in [n]$ as well as that such a cut can be computed in linear time. The same section discusses also how to extend the result to c-approximate m-cuts in trees. In Section 4.2, both results are generalized to tree-like graphs with a given tree decomposition and Section 4.3 discusses how approximate cuts can be used to construct exact cuts.

4.1 Approximate Cuts in Trees and Forests

To begin with, it is shown that every bounded-degree tree T allows a simple approximate cut of low width.

Lemma 4.1.
For every $n \in \mathbb{N}$ and every $m \in [n]$ the following holds. Every forest G on n vertices allows a simple approximate m-cut (B, W) with $e_G(B, W) \leq \Delta(G)$. A simple approximate m-cut in G whose width satisfies this bound can be computed in $\mathcal{O}(n)$ time.

Proof. First, it is shown that a cut with the desired properties exists and then it is discussed how to implement an algorithm computing such a cut. For the existence part, it is enough to prove the lemma for trees. Indeed, let G be a forest on n vertices that is not connected and fix an $m \in [n]$.

If $\Delta(G) \leq 1$, then it is easy to see that there is an exact m-cut (B, W) in G with $e_G(B, W) \leq \Delta(G)$ and this cut (B, W) satisfies all requirements. Otherwise, one can add edges to G to obtain a tree T with the same vertex set and the same maximum degree. Then, every cut (B, W) in T is also a cut in G and satisfies $E_G(B, W) \subseteq E_T(B, W)$.

Now, let $T = (V, E)$ be an arbitrary tree on n vertices and fix an integer $m \in [n]$. If $m = n$, set $B := V$ and $W := \emptyset$, which satisfy the requirements of the lemma. So, from now on, assume that $m < n$. Root T at an arbitrary vertex r and observe that then there are at least $m + 1$ vertices in the subtree rooted at r. Let k be the number of children of r, denote by $w_1, w_2, \ldots, w_k$ the k children of r, and, for each $h \in [k]$, let V_h be the vertex set of the subtree rooted at w_h. If there is an $\ell \in [k]$ with $|V_\ell| \geq m + 1$, consider only the subtree T' rooted at w_ℓ, refer to w_ℓ as the root r from now, and adjust k, w_h, and V_h for each $h \in [k]$ accordingly. Reapply this procedure until it is not possible anymore, i. e., until a subtree T' rooted at a vertex r is found, such that T' contains at least $m + 1$ vertices and $|V_h| \leq m$ for all $h \in [k]$. Note that r must have at least one child, as there are at least $m + 1$ vertices in the subtree rooted at r. Now, if there is an $\ell \in [k]$ with $|V_\ell| > \frac{1}{2}m$, define $B := V_\ell$ and $W := V \setminus B$. Then, the cut (B, W) cuts one edge in the original tree T. Otherwise, $|V_h| \leq \frac{1}{2}m$ for all $h \in [k]$ and $\sum_{h=1}^{k} |V_h| \geq m$ by the choice of r. Then, let ℓ be the largest integer in $[k]$ with $\sum_{h=1}^{\ell} |V_h| \leq m$ and set $B := \bigcup_{h=1}^{\ell} V_h$. As the sets $V_1, \ldots, V_\ell$ are pairwise disjoint, the choice of B satisfies $|B| \leq m$ as well as

$$\begin{aligned} |B| \; &> \; m - |V_{\ell+1}| \; \geq \; m - \tfrac{1}{2}m \; = \; \tfrac{1}{2}m && \text{if } \ell < k, \\ |B| \; &\geq \; |V(T')| - 1 \; \geq \; m \; > \; \tfrac{1}{2}m && \text{if } \ell = k. \end{aligned}$$

Defining $W := V \setminus B$ yields $e_T(B, W) \leq \ell \leq \deg(r) \leq \Delta(T)$, because exactly the edges $\{w_h, r\}$ with $h \in [\ell]$ are cut by (B, W).

Next, an implementation of an algorithm computing a simple approximate m-cut in a forest G with the required properties is discussed. As usual, the set B will be returned as an unordered list of vertices. If $m = n$, then it is trivial to compute the set $B = V$, i. e., the algorithm returns a list of all vertices in G. So from now on assume that $m < n$ and consider first the case when the input graph G is a tree T on n vertices. Without loss of generality, assume that T is given as an arborescence with root r as otherwise the algorithm contained in Lemma 2.33 can be applied. For each $v \in V$, let T_v be the subtree of T that is rooted at v and denote by n_v the number of vertices in the subtree rooted at v.

Claim 4.2.
There is an algorithm that computes the following in $\mathcal{O}(n)$ time:
(i) a vertex v^* with $n_{v^*} > m$ and $n_w \leq m$ for all children w of v^*, as well as
(ii) n_v for all descendants v of v^*.

Indeed, consider an algorithm that traverses T with a depth-first search starting at r and that computes n_v for each $v \in V$ when v turns black. The algorithm stops the traversal as soon as the first black vertex v^* with $n_{v^*} > m$ is reached. To do so, for each leaf $v \in V$, the algorithm sets $n_v = 1$ and, for each vertex $v \in V$ that is not a leaf of T, it computes

$$n_v \; = \; 1 + \sum_{w \text{ child of } v} n_w.$$

This works, since a vertex only turns black once all its children are black. The traversal itself takes $\mathcal{O}(n)$ time and the time to process one vertex $v \in V$ is proportional to $\deg(v)$. So the algorithm takes at most

$$\mathcal{O}(n) + \mathcal{O}\left(\sum_{v \in V} \deg(v)\right) \; = \; \mathcal{O}(n)$$

time to determine v^* and to compute n_v for all descendants of v^*. This completes the proof of Claim 4.2.

Now, the tree T_{v^*} has the same properties as the tree T' in the existence part of the proof and therefore, the algorithm can use the construction from above to determine ℓ children $w_1, \dots, w_\ell$ of v^* in $\mathcal{O}(\deg(v^*))$ time such that the set $B := \bigcup_{h\in[\ell]} V(T_{w_h})$ has the desired properties. Then, the algorithm computes the list of the vertices in the set B by traversing the subtrees T_{w_h} for all $h \in [\ell]$ and inserting the discovered vertices in the list for the set B. Computing the list of the vertices in the set B takes at most $\mathcal{O}(n)$ time and, hence, the complete algorithm runs in $\mathcal{O}(n)$ time.

Next, the case when the input graph is not connected is discussed. The algorithm does not add edges as done above in the proof of the existence part. Consider a forest G on n vertices and fix an $m \in [n]$. Denote by $T_1, \dots, T_k$ the components of G. Let ℓ be the largest integer in $[k] \cup \{0\}$ with $\sum_{h=1}^{\ell} |V(T_h)| < m$ and note that $\ell < k$. If $\ell = 0$, define $\tilde{B} := \emptyset$ and $\tilde{m} := 0$. Otherwise, define $\tilde{B} := \bigcup_{h\in[\ell]} V(T_h)$ and $\tilde{m} := |\tilde{B}|$. Now, apply the construction for trees to the tree $T_{\ell+1}$ with size-parameter $m' := m - \tilde{m}$ to obtain a simple approximate m'-cut (B', W') in $T_{\ell+1}$. Note that this is feasible as $m' \in [n']$ for $n' := |V(T_{\ell+1})|$. Define $B := \tilde{B} \,\dot\cup\, B'$, which satisfies

$$\tfrac{1}{2}m \;=\; \tfrac{1}{2}\tilde{m} + \tfrac{1}{2}m' \;\le\; \tilde{m} + \tfrac{1}{2}m' \;<\; |B| \;\le\; \tilde{m} + m' \;=\; m.$$

To implement this idea, one can use a depth-first search to determine the sizes of the components of G as well as a list of the vertices in T_h for each $h \in [\ell]$ in $\mathcal{O}(n)$ time according to Lemma 2.25 and then run the algorithm for trees, which takes at most $\mathcal{O}(n)$ additional time. □

Next, there is a quick, technical remark about the construction of the simple approximate cut, that will be important in Chapter 6 when proving Lemma 6.2.

Remark 4.3.
Note that, if $m < n$ and the algorithm contained in the previous proof is applied to a tree T with root r, then the computed simple approximate cut (B, W) will satisfy $r \in W$.

Our next aim is to strengthen the previous lemma to c-approximate m-cuts of small width. Considering a perfect ternary tree on n vertices, $m = \lfloor \frac{1}{2}n \rfloor$, and c tending to one, it is easy to see that the bound on the cut width of a c-approximate m-cut will have to depend on c. In other words, the closer c is to one, the more restrictive the bounds on the set B become but also the bound on the cut width will have to increase.

Before stating the next lemma about c-approximate m-cuts, let us discuss one idea for generalizing the previous lemma to c-approximate m-cuts. Consider a forest G on n vertices, fix an $m \in [n]$ and a $c \in [0, 1)$. If $c \le \frac{1}{2}$, then the size constraints of a c-approximate m-cut are less restrictive than the ones in the simple approximate m-cut and, hence, there is nothing to show. So assume that $c > \frac{1}{2}$, then one can iteratively find a set B with $cm \le |B| \le m$ by applying the following procedure. Start with $B := \emptyset$. As long as $|B| < cm$ construct a simple approximate m'-cut (B', W') in the forest $G - B$ with parameter $m' := m - |B|$ and then add the set B' to the set B. Clearly, the size of B will never exceed m and once the procedure stops, the cut (B, W) with $W = V(G) \setminus B$ is a c-approximate m-cut in G. Before each iteration $m - |B| > (1 - c)m$ is satisfied. Thus, more than $\frac{1-c}{2}m$ vertices are added to the set B in each iteration and the procedure stops after at most

$$\left\lceil \frac{cm}{\frac{(1-c)}{2}m} \right\rceil = \left\lceil \frac{2c}{1-c} \right\rceil$$

iterations. Furthermore, in each iteration, at most $\Delta(G)$ edges are cut additionally to the edges cut by the previous cut and, hence, a c-approximate m-cut of width at most $\left\lceil \frac{2c}{1-c} \right\rceil \Delta(G)$ is obtained. Using the algorithm for the simple approximate m-cut, that is contained in Lemma 4.1, it is easy to implement this procedure to run in $\mathcal{O}\left(\left\lceil \frac{2c}{1-c} \right\rceil n\right)$ time. The existence part is summarized in the following corollary.

Corollary 4.4.
For every $c \in [0,1)$, for every $n \in \mathbb{N}$, and for every $m \in [n]$, the following holds. Every forest G on n vertices allows a c-approximate m-cut (B,W) with $e_G(B,W) \leq \left\lceil \frac{2c}{1-c} \right\rceil \Delta(G)$.

The next lemma uses the same idea to construct a c-approximate m-cut, but the number of vertices in the set B is estimated more carefully to obtain a tighter bound on the number of cut edges. Furthermore, the algorithm runs faster as it reuses parameters that are computed during a preprocessing and ensures that after using a simple approximate cut in a vertex r the same vertex r will not be used for another simple approximate cut, where the vertex r refers to the root of the tree T' from the construction in the proof of Lemma 4.1.

Lemma 4.5 (extended version of Lemma 1.12).
For every $c \in [0,1)$, for every forest G on n vertices, and for every integer $m \in [n]$ the following holds.

a) If $c \neq 0$, there is a strict c-approximate m-cut (B,W) in G with $e_G(B,W) \leq \left\lceil \log_2\left(\frac{1}{1-c}\right)\right\rceil \Delta(G)$.

b) There is a c-approximate m-cut (B,W) in G with $e_G(B,W) \leq \left\lceil \log_2\left(\frac{1}{1-c}\right)\right\rceil \Delta(G)$.

In both cases, there is an algorithm that computes a cut with these properties in $\mathcal{O}(n)$ time, where the hidden constant in the running time depends neither on c nor on $\Delta(G)$.

The previous lemma states the bound on the cut width and the running time with their explicit dependance on c, as later, in Section 4.3 and in Section 5.2, Lemma 4.5 is applied with a value of c that depends on the number of vertices of the considered forest. Note that, as mentioned above, the closer c is to one, the closer the size of B is to m, but the bound on the number of cut edges increases also. For example, in Section 4.3 we study c-approximate m-cuts with c large enough to force $|B| = m$.

Proof of Lemma 4.5. Let $G = (V,E)$ be an arbitrary forest on n vertices and fix an integer $m \in [n]$. Observe first that, if $c = 0$, then the cut (B,W) with $B := \emptyset$ and $W := V$ has width zero and is a c-approximate m-cut in G, which can be computed in $\mathcal{O}(n)$ time. Hence, it suffices to prove Part a) as, for $c \neq 0$, Part b) follows directly from Part a).

So, let us now focus on Part a). Fix a $c \in (0,1)$. First, it is shown that a strict c-approximate m-cut in G exists and then it is discussed how to implement an algorithm following the construction. As in the proof of Lemma 4.1, for the existence part, it suffices to consider the case when G is a tree T as otherwise it is either trivial to find a cut with the desired properties, or one can add edges to G until G is connected without increasing its maximum degree. Also, if $m = n$, then the cut (B,W) with $B := V$ and $W := \emptyset$ satisfies all requirements. So, from now on assume that $m \in [n-1]$. In the following, it is argued that the procedure described in Algorithm 4.1 computes a strict c-approximate m-cut in T. Algorithm 4.1 uses the following notation. After rooting the tree T, for each $v \in V(T)$, the subtree of T that is rooted at v and where v is designated the root is denoted by T_v. As in Lemma 4.1, for $v \in V(T)$, the number of vertices in the tree T_v is denoted by n_v.

In the following, the while loops in Lines 7-18 and Lines 13-16 are referred to as the outer and inner while loop, respectively. Note that the preprocessing in Lines 1-6 is feasible. Indeed, Line 2 can always be executed as $m \leq n-1$ and the root r chosen in Line 1 satisfies $n_r = n$. If r does not satisfy the properties required for the vertex v^* in Line 2, then r has a child r' with $n_{r'} > m$. Again, if r' does not satisfy the properties required for the vertex v^* in Line 2, then r' has a child r'' with $n_{r''} > m$. This sequence must

Algorithm 4.1: Computes a strict c-approximate m-cut in a tree.

```
   Input: tree $T = (V, E)$ on $n$ vertices, integer $m \in [n-1]$, a real number $c \in (0, 1)$.
   Output: a strict $c$-approximate $m$-cut $(B, W)$ in $T$.
1  Choose an arbitrary vertex $r \in V$ and turn $T$ into an arborescence with root $r$;
2  Find a vertex $v^* \in V$ with $n_{v^*} > m$ and $n_w \leq m$ for all children $w$ of $v^*$
       and compute $n_w$ for all descendants $w$ of $v^*$;
3  ForEach $v$ in the subtree rooted at $v^*$ do
4      Sort the list of children $(w_1, w_2, \ldots, w_k)$ of $v$ such that $n_{w_1} \geq n_{w_2} \geq \cdots \geq n_{w_k}$;
5  EndFch
6  $B \leftarrow \emptyset, \quad v \leftarrow v^*$;
7  While $|B| \leq cm$ do
8      Let $k$ be the number of children of $v$ and let $(w_1, w_2, \ldots, w_k)$ be the list of children of $v$;
9      Let $\ell$ be the largest integer in $[k]$ with $\sum_{h=1}^{\ell} n_{w_h} \leq m - |B|$;
10     $B \leftarrow B \cup \left(\bigcup_{h=1}^{\ell} V(T_{w_h})\right)$;
11     If $\ell < k$ then
12         $w \leftarrow w_{\ell+1}$;
13         While $w \neq$ null and $n_w > m - |B|$ do
14             $v \leftarrow w$;
15             If $v$ has a child then let $w$ be the first child of $v$ else $w \leftarrow$ null;
16         Endw
17     Endif
18 Endw
19 Return $(B, V \setminus B)$;
```

stop with a vertex r^* with the properties required for the vertex v^* in Line 2 as no vertex is repeated in the sequence and the number of vertices in T is finite.

Next, some invariants of the outer while loop are stated. Before each execution of the outer while loop, the following holds:

(i) v is a descendant of v^*,

(ii) $B \cap V(T_v) = \emptyset$, as well as

(iii) $n_v \geq (m+1) - |B|$ and $n_w \leq m - |B|$ for every child w of v.

Furthermore, for $s \in \mathbb{N}$, if the outer while loop is executed s or more times, then

(iv) $\left(1 - \frac{1}{2^s}\right) m < |B| \leq m$ and

(v) $e_T(B, V \setminus B) \leq s\Delta(T)$

hold after the s^{th} execution of the outer while loop.

Due to Line 2 and Line 6, (i)-(iii) are satisfied before the first execution of the outer while loop. Now, fix an integer $s \geq 1$, such that the outer while loop is executed s or more times, and assume that (i)-(iii) are satisfied before the s^{th} execution of the outer while loop and that, if $s \geq 2$, (iv) and (v) are satisfied after the $(s-1)^{\text{st}}$ execution of the outer while loop. We will show now that the s^{th} execution of the outer while loop can be carried out, that (iv) and (v) are satisfied after the s^{th} execution of the outer while loop, and that (i)-(iii) are satisfied before the $(s+1)^{\text{st}}$ execution of the outer while loop if there is an $(s+1)^{\text{st}}$ execution of the outer while loop. Denote by B and v the state of the variables B and v in the algorithm before the s^{th} execution of the outer while loop, respectively, and by B' and v' the states of the variables B and v after the s^{th} execution of the outer while loop. As, in the s^{th} execution of the outer while loop in the algorithm, let k be the number of children of v, let $(w_1, \ldots, w_k)$ be the list of children of v, that is ordered according to Line 4, and let ℓ be the largest integer in $[k]$ with $\sum_{h=1}^{\ell} n_{w_h} \leq m - |B|$. Such an ℓ exists and is at least 1 for the following reason. On the one hand, $k \geq 1$ since $n_v \geq (m+1) - |B| \geq 2$ by (iii) and as $|B| \leq cm < m$, which implies that v has at least one child. On the other hand, (iii) implies that $n_{w_h} \leq m - |B|$ for all $h \in [k]$ and therefore $\ell \geq 1$. By (ii), all unions in Line 10 are disjoint and therefore $|B'| = |B| + \sum_{h=1}^{\ell} n_{w_h}$. Furthermore, the choice of ℓ in Line 9 ensures that $|B'| \leq m$. Now, if $n_{w_1} > \frac{1}{2}(m - |B|)$ then

$$|B'| \ > \ |B| + \tfrac{1}{2}(m - |B|) \ = \ \tfrac{1}{2}(m + |B|),$$

because $\ell \geq 1$ as argued before. Otherwise, $n_{w_h} \leq \frac{1}{2}(m - |B|)$ for all $h \in [k]$ as the list of children of v was ordered in Line 4 due to (i), and

$$|B'| \ > \ m - \tfrac{1}{2}(m - |B|) \ \geq \ \tfrac{1}{2}(m + |B|).$$

Consequently, $|B'| > \frac{1}{2}(m + |B|)$ holds in both cases. If $s = 1$, this suffices to show that (iv) is satisfied after the s^{th} execution of the outer while loop. If $s > 1$, then

$$|B'| \ > \ \tfrac{1}{2}(m + |B|) \ > \ \frac{1}{2}\left(m + \left(1 - \frac{1}{2^{s-1}}\right) m\right) \ = \ \left(1 - \frac{1}{2^s}\right) m,$$

as (iv) is satisfied after the $(s-1)^{\text{st}}$ execution of the outer while loop. Furthermore, Line 10 implies that $B' \setminus B = \bigcup_{h \in [\ell]} V(T_{w_h})$ and therefore the cut $(B' \setminus B, V(T_v) \setminus B')$ in the tree T_v cuts at most ℓ edges. So with (ii) it follows that every edge that is cut by $(B', V \setminus B')$ and not cut by $(B, V \setminus B)$ is incident to v. Consequently, (v) is satisfied after the s^{th} execution of the outer while loop.

If $\ell = k$, then $B' = B \,\dot{\cup}\, (V(T_v) \setminus \{v\})$ and (iii) implies that $|B'| \geq |B| + n_v - 1 \geq m > cm$. Therefore, if $\ell = k$, the s^{th} execution of the outer while loop is the last execution of the outer while loop and the proof of the invariants is complete. So, from now on, assume that $\ell < k$, i. e., Lines 11-17 are executed. The choice of ℓ in Line 9 implies that $n_{w_{\ell+1}} > m - |B'|$ and, hence, the inner while loop is executed at least once during the s^{th} execution of the outer while loop. Therefore, (iii) is satisfied before the $(s+1)^{\text{st}}$ execution of the outer while loop. Moreover, v' is a descendant of $w_{\ell+1}$, which shows that (i) and (ii) are satisfied before the $(s+1)^{\text{st}}$ execution of the outer while loop. This completes the proof of the invariants.

By (iv) and the condition on the outer while loop, the algorithm terminates with a strict c-approximate m-cut. Denote by s^* the number of executions of the outer while loop. By (iv), the final set B satisfies $|B| > \left(1 - \frac{1}{2^{s^*}}\right) m$. Furthermore,

$$\left(1 - \frac{1}{2^{\log_2\left(\frac{1}{1-c}\right)}}\right) m \ = \ \left(1 - \frac{1}{\frac{1}{1-c}}\right) m \ = \ (1 - (1-c))\, m \ = \ cm$$

and, hence, $s^* \leq \left\lceil \log_2 \left(\frac{1}{1-c}\right) \right\rceil$. Now, (v) shows that the desired bound on the width of the returned cut is satisfied.

Next, it is discussed how to implement Algorithm 4.1 to run in $\mathcal{O}(n)$ time, where the hidden constant depends neither on c nor on $\Delta(T)$. The case when $m = n$ is trivial and the case when the input graph is not connected is discussed later. As usual, the algorithm stores the set B as an unordered list. Every time the vertex set of a tree T_w is added to B in Line 10, the union is disjoint, as argued above. So, the algorithm can traverse the tree T_w and add all its vertices to the list storing the set B, which takes time proportional to n_w as T is an arborescence. While doing so, the algorithm keeps track of the size of the set B to check the condition in Line 7 in $\mathcal{O}(1)$ time and to compute numbers as $m - |B|$ in Line 9 and Line 13 in $\mathcal{O}(1)$ time. In total, at most $\mathcal{O}(n)$ time is needed for collecting the vertices that are put into the set B during the entire algorithm. Thus, in the following, the time needed for updating the set B is neglected.

As shown above, the vertex v is reset to one of its descendants $v' \neq v$ in the inner while loop in each execution of the outer while loop, except maybe for the last execution of the outer while loop. Therefore, the algorithm spends at most $\mathcal{O}(n)$ time for executing the inner while loop during the entire execution. Other than updating the set B in Line 10 and the time needed for the inner while loop, one execution of the outer while loop takes $\mathcal{O}(\deg(v))$ time, as the values n_{w_h} needed for Line 9 are known because of (i) and Line 2. Therefore, all executions of the outer while loop take at most $\mathcal{O}(\sum_{v\in V} \deg(v)) = \mathcal{O}(n)$ time. So, including the time for updating the set B and the time spent in all executions of the inner while loop, all executions of the outer while loop take at most $\mathcal{O}(n)$ time together.

Consider the preprocessing in Lines 1-6. Lemma 2.33 implies that Line 1 takes at most $\mathcal{O}(n)$ time and, in the proof of Lemma 4.1, it was argued that the procedure in Line 2 takes $\mathcal{O}(n)$ time, see Claim 4.2. So, to complete the proof of the running time, only Lines 3-5 need to be analyzed. If $\Delta(T)$ is constant, then clearly the sorting of all lists of children in Lines 3-5 can be done in $\mathcal{O}(n)$ time. However, it was claimed that the running time is independent of $\Delta(T)$. To achieve this, the algorithm uses the counting sort algorithm to sort all relevant lists of children simultaneously. By Lemma 2.19 it takes $\mathcal{O}(n)$ time to sort the $\sum_{v\in V(T_{v^*})}(\deg(v)-1) \leq 2n$ values n_w where w is a vertex whose parent is in T_{v^*}, since $n_w \in [n]$ for all considered values n_w. Afterwards, the algorithm splits the big sorted list into separate sorted lists, one for each vertex. This process requires each cell that contains a value n_w to know to which list it belongs, i. e., to know the parent of w. This way the algorithm uses only once the part of the counting sort algorithm that takes $\Theta(n)$ time. As this is the first time that an algorithm uses the counting sort algorithm to sort several lists simultaneously, we explain the details: The algorithm sets up an array A of n lists, where each list is initialized as an empty list. Then, it traverses T_{v^*} with a depth-first search that starts at v^* and when a vertex v turns black, it traverses the list of children $(w_1, w_2, \ldots, w_k)$ of v, inserts the child w_h and a pointer to v at the end of the list $A[n_{w_h}]$ for each $h \in [k]$, and resets the list of children of v to the empty list. After traversing T_{v^*}, the algorithm goes through the lists in the array A in the order given by A, starting with $A[1]$. When processing an entry consisting of a vertex w and a pointer to a vertex v, the algorithm inserts w in the beginning of the list of children of v. Note that this procedure sorts all lists of children of the vertices in T_{v^*} simultaneously, as the list $A[h]$ contains all vertices $w \in V(T_{v^*})$ with $n_w = h$ and a pointer to the parent of w. Initializing the array A takes $\mathcal{O}(n)$ time. As mentioned above, there are at most $2n$ values that need to be sorted, so filling them into the array A takes at most $\mathcal{O}(n)$ time and the sum of the lengths of the lists in A is at most $\mathcal{O}(n)$ in the end. Hence, to compute the sorted lists of children, the traversal of A takes at most $\mathcal{O}(n)$ time. Consequently, the application of the counting sort algorithm to implement Line 3-5 takes at most $\mathcal{O}(n)$ time.

To finish the proof, it is only missing to discuss how to implement the algorithm when the input graph is not connected. This can be done similarly as in the proof of Lemma 4.1: Fix $c \in (0,1)$. Consider a forest G on n vertices and fix $m \in [n]$. Let $T_1, \ldots, T_k$ be the components of G. Furthermore, let ℓ be the largest integer in $[k] \cup \{0\}$ with $\tilde{m} := \sum_{h=1}^{\ell} |V(T_h)| < m$ and note that $\ell < k$. If $\ell = 0$, set $\tilde{B} := \emptyset$, and otherwise set $\tilde{B} := \bigcup_{h=1}^{\ell} V(T_h)$. Then, the algorithm for trees can be applied to find a strict c-approximate m'-cut (B', W') in $T_{\ell+1}$ for $m' := m - \tilde{m}$. Set $B := \tilde{B} \,\dot{\cup}\, B'$. Then,

$$cm \;\leq\; c\tilde{m} + cm' \;\leq\; \tilde{m} + cm' \;<\; |B| \;\leq\; \tilde{m} + m' \;=\; m,$$

i. e., (B, W) is a strict c-approximate m-cut in G. The argument for the running time follows from Lemma 2.25, analogously to discussing the case when the input graph is not connected in the proof of Lemma 4.1. □

Proof of Lemma 4.1	Proof of Lemma 4.6	
tree $T = (V, E)$,	graph $G = (V, E)$,	tree decomposition $(T, \mathcal{X})$ with $T = (V_T, E_T)$ and $\mathcal{X} = (X^i)_{i \in V_T}$,
T is rooted, consider only the subtree rooted at r,	consider only the vertices in Y^{i^*},	T is rooted, consider only the subtree rooted in i^*,
$k =$ number of children of r, children of r: $w_1, \dots, w_k$,		$k =$ number of children of i^*, children of i^*: $j_1, \dots, j_k$.
sets $V_1, \dots, V_k \subseteq V$,	sets $Y_1, \dots, Y_k \subseteq V$,	
$V(T_r) = \{r\} \mathbin{\dot\cup} \dot{\bigcup}_{h \in [k]} V_h$.	$Y^{i^*} = X^{i^*} \mathbin{\dot\cup} \dot{\bigcup}_{h \in [k]} \tilde{Y}^{j_h}$.	

Table 4.1: Overview on the notation used in the proofs of Lemma 4.1 and Lemma 4.6.

4.2 Approximate Cuts in Tree-Like Graphs

This section generalizes the results discussed in the previous section to tree-like graphs with a given tree decomposition. As in the previous section, simple approximate cuts in graphs with a given tree decomposition are studied first and then the ideas are generalized to c-approximate cuts for any $c \in (0, 1)$. So the first aim of this section is to prove the following lemma about simple approximate cuts in tree-like graphs.

Lemma 4.6.
Let G be an arbitrary graph on n vertices and let $(T, \mathcal{X})$ be a tree decomposition of G of width at most $t - 1$. For every integer $m \in [n]$, there is a simple approximate m-cut (B, W) in G with $e_G(B, W) \leq t\Delta(G)$. Moreover, if $V(G) = [n]$ and the tree decomposition $(T, \mathcal{X})$ is provided as input, then such a cut can be computed in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time. The corresponding algorithm only requires the tree decomposition $(T, \mathcal{X})$ and the parameter m as input, but not the graph G.

Before proving this lemma, some ideas to generalize the results for trees to tree-like graphs with a given tree decomposition are discussed. Instead of only dealing with the vertex set of the input graph, the construction and the corresponding algorithm also have to deal with the node set of the tree decomposition, as the decomposition tree is needed to generalize certain properties of trees. Table 4.1 gives an overview on the notation and its corresponding version in the tree case, i. e., the proof of Lemma 4.1, and the notation in the general case, i. e., the proof of Lemma 4.6. In the latter, one column contains the sets and vertices from the input graph and the second one contains the notation and nodes that refer to the decomposition tree.

To prove the previous lemma and also the following lemma about c-approximate cuts for arbitrary values of $c \in (0, 1)$, clusters from the tree decomposition are used to split up the input graph, instead of single vertices, similarly to Lemma 2.16. The resulting disjoint parts are then combined to construct the set B for the desired cut. Here, Proposition 3.7 from Section 3.1.3 will be useful for finding a node i whose removal creates disjoint parts of suitable sizes. Recall the following notation from Section 3.1.3: Consider a graph $G = (V, E)$ and a tree decomposition $(T, \mathcal{X})$ of G with $T = (V_T, E_T)$ and $\mathcal{X} = (X^i)_{i \in V_T}$.

Assume that T is rooted at an arbitrary node r and recall that, for every node $i \in V_T \setminus \{r\}$, the parent of i in T is denoted by $p(i)$. As in Section 3.1.3 let

$$\begin{aligned} Y^i &= \bigcup_{j \text{ descendant of } i} X^j && \text{for every } i \in V_T, \\ \tilde{Y}^i &= Y^i \setminus X^{p(i)} && \text{for every } i \in V_T \setminus \{r\}, \\ \tilde{Y}^r &= Y^r, \end{aligned}$$

as well as $y_i = |Y^i|$ and $\tilde{y}_i = |\tilde{Y}^i|$ for every $i \in V_T$.

Proposition 4.7 (Proposition 3.7 repeated).
For every graph G and every tree decomposition $(T, \mathcal{X})$ of G, where $T = (V_T, E_T)$ is a rooted tree and $\mathcal{X} = (X^i)_{i \in V_T}$, the following holds for the sets Y^i and $\tilde{Y}^i$ as defined above.

a) For every node $i \in V_T$,

$$Y^i = X^i \mathbin{\dot\cup} \left(\dot{\bigcup_{j \text{ child of } i}} \tilde{Y}^j \right).$$

b) For every node $i \in V_T$ with children $j_1, j_2, \dots, j_k$ and every partition $X^i = Z_1 \mathbin{\dot\cup} Z_2$,

$$E_G(Z_1, Z_2, \tilde{Y}^{j_1}, \tilde{Y}^{j_2}, \dots, \tilde{Y}^{j_k}, V(G) \setminus Y^i) \subseteq E_G(i).$$

c) If $V(G) = [n]$, then one can compute y_i and $\tilde{y}_i$ for every $i \in V_T$, all together in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time.

In the proof of Lemma 4.1, it was sufficient to use the vertices in the subtrees rooted at the children of some vertex r when constructing the black set and the vertex r itself was never used (except when $m = n$, i. e., the black set is allowed to contain all vertices). This will not work anymore for tree-like graphs: First, it can happen that the node i^* of the decomposition tree, that is chosen in the proof of Lemma 4.6 and corresponds to the vertex r in the tree case, does not have any children. Second, when disregarding the vertices in X^{i^*} it could be that the clusters associated with the nodes in the subtree rooted at i^* all together do not contain enough vertices for the set B, even when a node i^* with $y_{i^*} > m$ is chosen. Therefore, in some cases, the vertices in X^{i^*} are needed to construct the set B. Moreover, as the decomposition tree is always connected, it is not needed to consider separately the case when the input graph is not connected.

Proof of Lemma 4.6. Let $G = (V, E)$ be an arbitrary graph on n vertices and let $(T, \mathcal{X})$ be a tree decomposition of G of width at most $t - 1$. Furthermore, let $T = (V_T, E_T)$, let $\mathcal{X} = (X^i)_{i \in V_T}$, and fix some integer $m \in [n]$. Root T in an arbitrary node r and define Y^i, $\tilde{Y}^i$, y_i, and $\tilde{y}_i$ for every $i \in V_T$ as done before Proposition 4.7. Note that $y_r = n \geq m$ as (T1) implies $Y^r = V$. If r has a child i with $y_i \geq m$, then consider only the subtree of T that is rooted at i. Reapply this procedure until a node i^* is found with $y_{i^*} \geq m$ and $y_j < m$ for all children j of i^*. Next, the set B for the desired cut is constructed by using only vertices in Y^{i^*}. To ensure that only few edges are cut, the partition

$$Y^{i^*} = X^{i^*} \mathbin{\dot\cup} \left(\dot{\bigcup_{j \text{ child of } i^*}} \tilde{Y}^j \right)$$

from Proposition 4.7a) and the property in Proposition 4.7b) are used. Let k be the number of children of i^* and, if $k > 0$, denote by $\{j_1, \dots, j_k\}$ the set of children of i^*. In the following, a subset J of the children of i^* and a set $Z \subseteq X^{i^*}$ are defined. Furthermore, let $B := Z \mathbin{\dot\cup} \left(\dot\bigcup_{j \in J} \tilde{Y}^j \right)$ and $W := V \setminus B$.

Note that by Proposition 4.7a) the unions in the definition of the set B are indeed disjoint as well as that Proposition 4.7b) implies that the cut (B, W) cuts only edges in $E_G(i^*)$. Hence, the width of (B, W) is at most $t\Delta(G)$, as desired. Next, the sets J and Z are defined and it is shown that the resulting cut (B, W) is a simple approximate m-cut in G.

Case 1: $k = 0$, i. e., i^* has no children. In this case, $|X^{i^*}| = |Y^{i^*}| \geq m$ and therefore a set $Z \subseteq X^{i^*}$ of size m exists. Furthermore, let $J = \emptyset$. Clearly, (B, W) is a simple approximate m-cut in G.

Case 2: $k > 0$. The choice of i^* implies that $\tilde{y}_{j_h} \leq y_{j_h} < m$ for all $h \in [k]$.

Case 2a: There is a child j_ℓ of i^* with $\tilde{y}_{j_\ell} > \frac{1}{2}m$. In this case, define $J := \{j_\ell\}$ and $Z := \emptyset$. Note that the corresponding cut (B, W) is a simple approximate m-cut in G.

Case 2b: $\tilde{y}_{j_h} \leq \frac{1}{2}m$ for all $h \in [k]$. Then, let ℓ be the largest integer in $[k]$ with $\sum_{h\in[\ell]} \tilde{y}_{j_h} \leq m$ and define $J := \{j_h \colon h \in [\ell]\}$. If $\ell < k$, let $Z := \emptyset$ and otherwise, i. e., if $\ell = k$, choose a set $Z \subseteq X^{i^*}$ of size $m - \sum_{j\in J} \tilde{y}_j$. In the latter case, such a set Z exists as $m \leq y_{i^*} = |X^{i^*}| + \sum_{j\in J} \tilde{y}_j$ by Proposition 4.7a). If $\ell = k$, then $|B| = m$. Otherwise, by the choice of ℓ

$$|B| \;=\; \sum_{h\in[\ell]} \tilde{y}_{j_h} \;>\; m - \tilde{y}_{j_{\ell+1}} \;\geq\; \tfrac{1}{2}m.$$

Therefore, the cut (B, W) is a simple approximate m-cut in G.

This completes the existence part of the proof. Next, it is discussed how to compute a simple approximate m-cut when given $(T, \mathcal{X})$ as input and the assumption $V(G) = [n]$ is satisfied. To store the set B, the algorithm uses a binary array A_B of length n, that is initialized with zeros. In the end of the algorithm, an entry in A_B is one if and only if the corresponding vertex of G is in the returned set B.

First, the algorithm creates the array A_B and initializes it with zeros, which takes $\mathcal{O}(n)$ time. Then, it roots T at an arbitrary node r and converts T into an arborescence with root r, which takes $\mathcal{O}(n_T)$ time due to Lemma 2.33. Then, it applies the algorithm contained in Proposition 4.7c) to find a node in T with the properties of node i^* from the construction above. As the procedure used to compute the values y_i and $\tilde{y}_i$ is a depth-first search that processes the nodes when they turn black, the algorithm can stop as soon as it found the first node i with $y_i \geq m$. Then, this node i has the same properties as the node i^* used in the construction above. Hence, computing the node i^* takes $\mathcal{O}(\|(T, \mathcal{X})\|)$ time. Since the values y_j and $\tilde{y}_j$ are known for all descendants of i^* as they turned black before i^*, it takes $\mathcal{O}(\deg_T(i^*)) = \mathcal{O}(n_T)$ time to determine the set J. Before computing the set Z the algorithm adds the vertices in $\tilde{Y}^j$ to the set B for each $j \in J$, i. e., it sets the corresponding indices of A_B to one. To do so, for each $j \in J$, the algorithm goes through the subtree of T rooted at j and sets the entries of all vertices in the clusters of the nodes discovered there to one. Note that then, an entry of A_B is set to one if and only if the corresponding vertex is in a set Y^j with $j \in J$. Next, the algorithm goes through the cluster X^{i^*} and sets all corresponding entries in A_B to zero. Then, an entry in A_B is one if and only if the corresponding vertex is in a set $\tilde{Y}^j$ with $j \in J$. If $Z \neq \emptyset$ in the above construction, the algorithm goes on and sets the entries in A_B that correspond to vertices in the set Z to one, where the set Z can be constructed greedily by traversing the list of the cluster X^{i^*} once more since $\dot{\bigcup}_{j\in J}\tilde{Y}^j$ and X^{i^*} are disjoint by Proposition 4.7a). All together, collecting the vertices in the set B takes at most $\mathcal{O}(\|(T, \mathcal{X})\|)$ time. Therefore, the running time of the algorithm is $\mathcal{O}(n + n_T + \|(T, \mathcal{X})\|) = \mathcal{O}(\|(T, \mathcal{X})\|)$. □

Next, the previous lemma is generalized to c-approximate m-cuts in tree-like graphs with a given tree decomposition for any $c \in [0, 1)$. The strategy is similar to the one used in the previous section for trees.

There, it was also discussed that a repeated application of simple approximate cuts gives a c-approximate cut in a tree, but the running time depends on c. This idea works in the same way for tree-like graphs with a given tree decomposition, when computing the induced tree decomposition in order to apply a simple approximate cut to a subgraph of the input graph, see also Proposition 2.31b). However, to achieve a running time that is independent of c, as done in the next lemma, one needs to avoid to compute several induced tree decompositions. Algorithm 4.2 in the proof of the next lemma is more involved than its tree-version, Algorithm 4.1 in the proof of Lemma 4.5, as for example not in every iteration the considered node i has children and sometimes it is necessary to use vertices in the set X^i when constructing the black set for the cut.

Lemma 4.8 (extended version of Lemma 1.13).
For every $c \in [0,1)$, for every graph G on n vertices, for every tree decomposition $(T,\mathcal{X})$ of G of width $t-1$, and for every integer $m \in [n]$, the following holds.

a) If $c \neq 0$, there is a strict c-approximate m-cut (B,W) in G with $e_G(B,W) \leq \left\lceil \log_2\left(\frac{1}{1-c}\right)\right\rceil t\Delta(G)$.

b) There is a c-approximate m-cut (B,W) in G with $e_G(B,W) \leq \left\lceil \log_2\left(\frac{1}{1-c}\right)\right\rceil t\Delta(G)$.

Moreover, if $V(G)=[n]$, then, in both cases, such a cut can be computed in $\mathcal{O}(\|(T,\mathcal{X})\|)$ time, where the hidden constant does not depend on c. The corresponding algorithm only requires the tree decomposition $(T,\mathcal{X})$ and the parameters c and m as input, but not the graph G.

Proof. Let $G=(V,E)$ be an arbitrary graph on n vertices and let $(T,\mathcal{X})$ be a tree decomposition of G with $T=(V_T,E_T)$ and $\mathcal{X}=(X^i)_{i\in V_T}$. Denote by $t-1$ the width of $(T,\mathcal{X})$ and let $n_T := |V_T|$. As in the proof of Lemma 4.5, when $c=0$, the cut (B,W) with $B=\emptyset$ and $W=V$ has width zero and is a c-approximate m-cut in G. If $V(G)=[n]$, then the set B of the cut (B,W) can be computed in $\mathcal{O}(\|(T,\mathcal{X})\|)$ time, even if G is not given as input. Hence, it suffices to prove Part a) as, for $c \neq 0$, Part b) follows directly from Part a).

To prove Part a), fix an integer $m \in [n]$ and some $c \in (0,1)$. To obtain the set B, the procedure described in Algorithm 4.2 is applied. The idea behind it is to add vertices to the set B by applying Lemma 4.6 to $G[V\setminus B]$ repeatedly until the set B has the desired size. There is no need to compute the induced tree decomposition for $G[V\setminus B]$, as the order of the nodes i of T, that are used in the implicit applications of Lemma 4.6, is chosen in a way such that the subtree rooted at each node i and the corresponding clusters form a tree decomposition of a suitable subgraph of G, whose vertex set is disjoint from the current set B.

First, observe that Line 2 is feasible since $Y^r=V$ due to (T1) and, hence, $y_r=n\geq m$. In the following, the while loops in Lines 7-25 and Lines 20-23 are called the outer while loop and the inner while loop, respectively. Next, some invariants of the outer while loop are stated and will be proved in what follows. At the beginning of each execution of the outer while loop,

(i) $B\cap Y^i=\emptyset$ as well as

(ii) $y_i \geq m-|B|$ and $y_j < m-|B|$ for each child j of i.

To state two more invariants, denote by B_s and i_s the set B and the node i after the s^{th} execution of the outer while loop, respectively. Furthermore, set $B_0=\emptyset$ and let i_0 be the node i^* computed in Line 2, which are the states of B and i before the first execution of the outer while loop, respectively. For every $s\geq 1$ such that the outer while loop was executed at least s times, the following holds:

(iii) $\left(1-\frac{1}{2^s}\right)m < |B_s| \leq m$,

(iv) $e_G(B_s, V\setminus B_s) \leq st\Delta(G)$, and

(v) if the s^{th} execution of the outer while loop was not the last execution, then i_s is a descendant of i_{s-1}.

```
Algorithm 4.2: Computes a strict $c$-approximate $m$-cut in a tree-like graph.
Input: a tree decomposition $(T, \mathcal{X})$ of a graph $G$ on $n$ vertices, an integer $m \in [n]$, and a real
       number $c \in (0,1)$.
Output: a strict $c$-approximate $m$-cut $(B, W)$ in $G$.
1  Root $T$ in an arbitrary node $r$ and turn $T$ into an arborescence;
2  Find a node $i^* \in V(T)$ with $y_{i^*} \geq m$ and $y_j < m$ for all children $j$ of $i^*$
       and compute $y_i$ and $\tilde{y}_i$ for all descendants $i$ of $i^*$;
3  ForEach $i$ in the subtree rooted at $i^*$ do
4      Sort the list of children $(j_1, j_2, \ldots, j_k)$ of $i$ such that $\tilde{y}_{j_1} \geq \tilde{y}_{j_2} \geq \cdots \geq \tilde{y}_{j_k}$;
5  EndFch
6  $B \leftarrow \emptyset, \quad i \leftarrow i^*$;
7  While $|B| \leq cm$ do
8      Let $k$ be the number of children of $i$ and let $(j_1, j_2, \ldots, j_k)$ be the list of children of $i$;
9      If $k \geq 1$ then
10         Let $\ell$ be the largest integer in $[k]$ with $\sum_{h=1}^{\ell} \tilde{y}_{j_h} \leq m - |B|$;
11     Else
12         $\ell \leftarrow 0$;
13     Endif
14     $B \leftarrow B \cup \left( \bigcup_{h=1}^{\ell} \tilde{Y}^{j_h} \right)$;
15     If $\ell = k$ then
16         Let $Z$ be a subset of $X^i$ with $|Z| = m - |B|$;
17         $B \leftarrow B \cup Z$;
18     Else
19         $j \leftarrow j_{\ell+1}$;
20         While $j \neq$ null and $y_j \geq m - |B|$ do
21             $i \leftarrow j$;
22             If $i$ has a child then let $j$ be the first child of $i$ else $j \leftarrow$ null;
23         Endw
24     Endif
25 Endw
26 Return $(B, V \setminus B)$;
```

Clearly (i) and (ii) hold before the first execution of the outer while loop. For an arbitrary integer $s \geq 1$ such that the outer while loop is executed at least s times, assume that (i)-(ii) hold at the beginning of the s^{th} execution of the outer while loop and, if $s \geq 2$, that (iii)-(v) are satisfied for $s-1$. We will show that the s^{th} execution of the outer while loop can be carried out and that, if it is not the last execution, (i)-(ii) hold after the s^{th} execution, i. e., at the beginning of the $(s+1)^{\text{st}}$ execution of the outer while loop. Moreover, we will show that (iii)-(v) are satisfied for s. As in the algorithm, let k be the number of children of i_{s-1}, let $j_1, j_2, \ldots, j_k$ be the children of i_{s-1}, and denote by ℓ the value determined in Lines 9-13 during the s^{th} execution of the outer while loop. If Lines 16-17 are executed during the s^{th} execution of the outer while loop, let Z_s be the set Z computed in Line 16, and otherwise define

$Z_s = \emptyset$. The set B is only modified in Line 14 and Line 17, which gives

$$B_s = \begin{cases} B_{s-1} \cup Z_s \cup \left(\bigcup_{h=1}^{\ell} \tilde{Y}^{j_h} \right) & \text{if } k \geq 1, \\ B_{s-1} \cup Z_s & \text{if } k = 0. \end{cases} \tag{4.1}$$

The set B_{s-1} is disjoint from the sets that are added to it in the previous equation, because (i) holds at the beginning of the s^{th} execution of the outer while loop and $Z_s \subseteq X^{i_{s-1}} \subseteq Y^{i_{s-1}}$ as well as $\tilde{Y}^{j_h} \subseteq Y^{i_{s-1}}$ for every $h \in [k]$. Also, if $k \geq 1$, Proposition 4.7a) implies that the sets $Z_s \subseteq X^{i_{s-1}}$ and $\tilde{Y}^{j_h}$ for $h \in [k]$ are pairwise disjoint. Hence, all unions in (4.1) are disjoint unions and $|B_s| = |B_{s-1}| + |Z_s| + \sum_{h=1}^{\ell} \tilde{y}_{j_h}$. Moreover, since (ii) is satisfied at the beginning of the s^{th} execution of the outer while loop by assumption, Proposition 4.7a) implies that $m - |B_{s-1}| \leq y_{i_{s-1}} = |X^{i_{s-1}}| + \sum_{h=1}^{k} \tilde{y}_{j_h}$. Therefore, if Line 16 is reached in the s^{th} execution of the outer while loop, it is feasible. The choice of ℓ and Z_s together with (4.1) imply that $|B_s| \leq m$. Next, a few cases are examined to obtain a lower bound on the size of B_s.

Case 1: Lines 16-17 are executed in the s^{th} execution of the outer while loop, i. e., $k = 0$ or $k = \ell > 1$. Then, $|B_s| = m$ due to Line 17. Clearly, (iii) is then satisfied after the s^{th} execution of the outer while loop. Furthermore, the s^{th} execution was the last execution of the outer while loop.

Case 2: Lines 16-17 are not executed in the s^{th} execution of the outer while loop. Then, $Z_s = \emptyset$, $k \geq 1$, and $\ell < k$. Furthermore, ℓ must be at least 1 as (ii) is satisfied at the beginning of the s^{th} execution of the outer while loop.

Case 2a: $\tilde{y}_{j_1} > \frac{1}{2}(m - |B_{s-1}|)$. Then $|B_s| > |B_{s-1}| + \frac{1}{2}(m - |B_{s-1}|) \geq \frac{1}{2}(m + |B_{s-1}|)$ since $\ell \geq 1$.

Case 2b: $\tilde{y}_{j_1} \leq \frac{1}{2}(m - |B_{s-1}|)$. Then, $\tilde{y}_{j_h} \leq \frac{1}{2}(m - |B_{s-1}|)$ for all $h \in [k]$, because i_{s-1} is a descendant of $i_0 = i^*$ by (v) and, hence, the list of children of i_{s-1} has been sorted in Line 4. Since $\ell < k$ in Case 2, it follows that $|B_s| > m - \tilde{y}_{j_{\ell+1}} \geq m - \frac{1}{2}(m - |B_{s-1}|) \geq \frac{1}{2}(m + |B_{s-1}|)$.

Consequently, $|B_s| > \frac{1}{2}(m + |B_{s-1}|)$ holds in Case 2. If $s = 1$, this suffices to show that (iii) is satisfied for s. If $s > 1$, then (iii) is satisfied for $s - 1$ and implies that

$$|B_s| > \tfrac{1}{2}(m + |B_{s-1}|) > \frac{1}{2}\left(m + \left(1 - \frac{1}{2^{s-1}}\right) m\right) = \left(1 - \frac{1}{2^s}\right) m,$$

i. e., (iii) is satisfied for s.

This completes the case analysis and shows that (iii) is satisfied for s. Moreover, (4.1), $Z_s \subseteq X^{i_{s-1}}$, and Proposition 4.7b) imply that

$$e_G(B_s \setminus B_{s-1}, V \setminus (B_s \setminus B_{s-1})) \leq e_G(i_{s-1}) \leq t\Delta(G)$$

and, hence, invariant (iv) is satisfied for s, as it was satisfied for $s - 1$ by assumption if $s \geq 2$.

Furthermore, in Case 1, as mentioned above, the s^{th} execution of the outer while loop is the last execution of the outer while loop and, hence, there is nothing to show for (i), (ii), and (v). So assume that the s^{th} execution is not the last execution of the outer while loop, i. e., Case 2 applies and $k > \ell \geq 1$. Then, Lines 19-23 are executed and Line 19 is feasible. The choice of ℓ in Line 10 implies that $y_j \geq \tilde{y}_j > m - |B|$ holds after Line 19 is executed. Therefore, the inner while loop is executed at least once during the s^{th} execution of the outer while loop and (ii) is satisfied after the s^{th} execution of the outer while loop since i_{s-1} is a descendant of $i_0 = i^*$ by (v) and, hence, for each descendant of i_{s-1} the list of children has been sorted in Line 4. Moreover, this implies that i_s is a descendant of $j_{\ell+1}$ and (v) is satisfied for s. Hence, $Y^{i_s} \subseteq Y^{j_{\ell+1}} \subseteq \tilde{Y}^{j_{\ell+1}} \cup X^{i_{s-1}}$ and to show that (i) is satisfied after the s^{th} execution of the outer while loop, it suffices to show that $(\tilde{Y}^{j_{\ell+1}} \cup X^{i_{s-1}}) \cap B_s = \emptyset$. This is indeed satisfied since Proposition 4.7a), (4.1), and $Z_s = \emptyset$ due to Case 2 imply that

$$(\tilde{Y}^{j_{\ell+1}} \cup X^{i_{s-1}}) \cap B_s \;\subseteq\; (\tilde{Y}^{j_{\ell+1}} \cup X^{i_{s-1}}) \cap B_{s-1} \;\subseteq\; Y^{i_{s-1}} \cap B_{s-1} \;=\; \emptyset,$$

where the last equality holds because (i) is satisfied at the beginning of the s^{th} execution of the outer while loop by assumption. This completes the proof of the invariants and shows that every step can be executed.

Note that, in the previous paragraph, it was also shown that, in the s^{th} execution of the outer while loop, the node i_s is chosen to be a descendant of a child j of i_{s-1} such that $\tilde{Y}^j$ was not added to the set B in Line 14. This also implies that, for every child j of i_{s-1} such that $\tilde{Y}^j$ is added to the set B in Line 14 during the s^{th} execution of the outer while loop, the node i will never be reset to a descendant of j in the inner while loop.

By (iii) and the condition on the outer while loop, the algorithm terminates and returns a strict c-approximate m-cut. Denote by s^* the number of executions of the outer while loop. Then, (iii) implies that the final set B satisfies $|B| > \left(1 - \frac{1}{2^{s^*}}\right) m$. Furthermore,

$$\left(1 - \frac{1}{2^{\log_2\left(\frac{1}{1-c}\right)}}\right) m \;=\; \left(1 - \frac{1}{\frac{1}{1-c}}\right) m \;=\; (1 - (1-c))\, m \;=\; cm$$

and hence $s^* \leq \left\lceil \log_2\left(\frac{1}{1-c}\right) \right\rceil$, which together with (iv) shows that the desired constraint on the width of the returned cut is satisfied.

Next, it is discussed how to implement Algorithm 4.2 to achieve the desired running time. So from now on, assume that $V = [n]$. Similarly to the algorithm in the proof of Lemma 4.6, a binary array A_B of length n is used to store the set B. The array A_B is initialized with zeros, which takes $\mathcal{O}(n)$ time. An entry in A_B is set to one if and only if the corresponding vertex of G is in the set B. Therefore, it is important that the vertices of G are labeled with $1, 2, \dots, n$. Using the notation from above, during the s^{th} execution of the outer while loop, in Line 14, the algorithm needs to add the set $\bigcup_{h=1}^{\ell} \tilde{Y}^{j_h}$ to B, where j_h is a child of the node i_{s-1} for every $h \in [\ell]$. So consider one iteration where $\ell \geq 1$. To add the vertices in $\bigcup_{h=1}^{\ell} \tilde{Y}^{j_h}$ to the set B, i. e., to set the corresponding entries in A_B to one, for each $h \in [\ell]$, the algorithm goes through the subtree rooted at j_h and sets the entries of all vertices in the clusters of the nodes discovered there to one. So far, an entry of A_B is set to one in the s^{th} execution of the outer while loop if and only if the corresponding vertex is in $\bigcup_{h\in[\ell]} Y^{j_h} \subseteq Y^{i_{s-1}}$. Next, the algorithm goes through the list representing the cluster $X^{i_{s-1}}$ and sets all corresponding entries in A_B to zero. While doing so, no entry of A_B was only set back to zero in the s^{th} execution of the outer while loop, since (i) ensures that none of the entries corresponding to a vertex in $X^{i_{s-1}} \subseteq Y^{i_{s-1}}$ was one in the beginning of the s^{th} execution of the outer while loop. Since i_{s-1} is the parent of j_h for every $h \in [\ell]$, an entry of A_B was set to one during the s^{th} execution of the outer while loop if and only if the corresponding vertex is in the set $\bigcup_{h\in[\ell]} \tilde{Y}^{j_h}$. With the observation above that, in Line 21, the node i is never reset to a descendant of a node j for which the set $\tilde{Y}^j$ was added to B in Line 14, it follows that each set X^i with $i \in V_T$ is traversed at most once for the addition of its vertices to B in Line 14. Other than in Line 14, the set B can be modified in Line 17. If executed, Lines 16-17 together take $\mathcal{O}(|X^{i_{s-1}}|)$ time when constructing the set Z greedily by going through the list of the cluster $X^{i_{s-1}}$, which is disjoint from the current set B. Hence, all modifications on the set B, or more precisely on the array A_B, and finding a suitable set Z in Line 16, if executed, together take at most $\mathcal{O}(n + \|(T, \mathcal{X})\|) = \mathcal{O}(\|(T, \mathcal{X})\|)$ time, during the entire algorithm. In the following, we consider Line 16, in which the set Z is computed, to belong to the steps of updating the set B and we do not consider the time needed to update the set B when estimating the time consumed by the other lines.

Each node of T is considered at most once during all executions of the inner while loop throughout the entire algorithm, because of (v). So, at most $\mathcal{O}(n_T)$ time is needed for the inner while loop during the entire algorithm. Furthermore, as the time needed to execute Lines 8-13 for a node i is proportional to $\mathcal{O}(\deg_T(i)+1)$, the time needed for Lines 8-13 during the entire algorithm is $\mathcal{O}(n_T)$. Consequently, the time needed for all iterations of the outer while loop, except for updating the set B, is $\mathcal{O}(n_T)$.

Next, the time needed for the preprocessing in Lines 1-6 is analyzed. Line 1 takes $\mathcal{O}(n_T)$ time due to Lemma 2.33 and Line 2 can be implemented similarly to the algorithm in Lemma 4.6. Therefore, Lines 1-2 take $\mathcal{O}(\|(T,\mathcal{X})\|)$ time. Since $\tilde{y}_j \in [n]$ for every $j \in V_T$, the counting sort algorithm from Lemma 2.19 can be used to implement Lines 3-5. As there are at most $\sum_{i\in V_T} \deg_T(i) \le 2n_T$ values in all lists of children together, this takes $\mathcal{O}(n_T+n) = \mathcal{O}(\|(T,\mathcal{X})\|)$ time to sort all considered lists of children simultaneously. The details are analogous to the application of the counting sort algorithm in the proof of Lemma 4.5, where it is explained with more details.

All in all, the preprocessing takes $\mathcal{O}(\|(T,\mathcal{X})\|)$ time, all modifications on the set B take $\mathcal{O}(\|(T,\mathcal{X})\|)$ time, and all other steps take $\mathcal{O}(n_T)$ time, which sums up to a running time of $\mathcal{O}(\|(T,\mathcal{X})\|)$. □

In the proof of the previous lemma, the counting sort algorithm is really needed to sort all lists of children simultaneously, even when the considered graph has bounded degree and the provided tree decomposition is nonredundant. Indeed, consider a graph G on n vertices and a tree decomposition $(T,\mathcal{X})$ of G of width $t-1$. Furthermore consider an integer $m \in [n]$ and assume that m is almost n, such that it can happen that almost all adjacency lists of T need to be sorted when applying Algorithm 4.2 to $(T,\mathcal{X})$ with parameter m and some $c \in (0,1)$. Applying the counting sort algorithm to each node separately, $\theta(n)$ time is needed for each node, which gives a total time of $\theta(n \cdot |V(T)|)$ and this can be asymptotically larger than $\|(T,\mathcal{X})\|$. For example, if the width $t-1$ is constant and $(T,\mathcal{X})$ is nonredundant, then $\|(T,\mathcal{X})\| \le |V(T)| \cdot t \le nt$ by Proposition 2.32a), but $|V(T)|$ can still be as large as $\Omega(n)$ and indeed must be so large as $\frac{n}{t}$ clusters are needed due to (T1). Moreover, we cannot assume that the tree T has bounded degree, even if the graph G has bounded degree. If the tree T had bounded degree, then each list of children could be sorted in constant time by applying a standard sorting technique that sorts n' values in $\mathcal{O}(n' \log n')$ time. To demonstrate that the tree T can have large maximum degree even when the graph G has bounded degree, consider the following example: Let G be a bounded-degree forest on n vertices that consists of $\Theta(\sqrt{n})$ components. A nonredundant tree decomposition of G of minimum width can be obtained from taking a nonredundant tree decomposition of minimum width of each component and joining them in the following way. Choose one node i from a decomposition tree of an arbitrary component and for all other components G', join i to an arbitrary node of the decomposition tree of G'. Then, the degree of i in the resulting tree decomposition of G is $\Theta(\sqrt{n})$.

4.3 Constructing Exact Cuts Through Approximate Cuts

In the previous two sections, it was shown that bounded-degree trees and bounded-degree tree-like graphs admit approximate cuts of small width. Consider a c-approximate m-cut (B,W) in a graph G. The closer c is to one, the closer $|B|$ is to m. In particular, when c tends to one, the set B is forced to have size exactly m. Therefore, one can use approximate cuts with certain parameters to construct exact cuts. Recall that a cut (B,W) is an (exact) m-cut if $|B| = m$. Consider a graph G on n vertices and fix an $m \in \mathbb{N}$ with $m \le \frac{1}{2}n$. Set $c := 1 - \frac{1}{m}$ and let (B,W) be a strict c-approximate m-cut in G. Then,

$m - 1 = cm < |B| \leq m$ implies that $|B| = m$ and (B, W) is an exact m-cut. The restriction $m \leq \frac{1}{2}n$ can be circumvented because, if $n > m > \frac{1}{2}n$, a strict $(1 - \frac{1}{n-m})$-approximate $(n-m)$-cut (B, W) is an exact $(n-m)$-cut and switching the sets B and W yields an exact m-cut. This construction together with Lemma 4.5 and Lemma 4.8 and the computation

$$\left\lceil \log_2\left(\frac{1}{1-c}\right)\right\rceil \;=\; \left\lceil \log_2\left(\frac{1}{1-(1-\frac{1}{m})}\right)\right\rceil \;\leq\; \log_2(m) + 1 \;=\; \log_2(2m)$$

immediately gives the following corollaries. Note that here it is important that the running times in Lemma 4.5 and Lemma 4.8 do not depend on c.

Corollary 4.9.
For every forest G on n vertices and every $m \in [n]$, the following holds. There is an m-cut (B, W) in G with

$$e_G(B, W) \;\leq\; \min\{\log_2(2m), \log_2(2(n-m))\}\,\Delta(G) \;\leq\; \log_2(n)\Delta(G).$$

A cut with these properties can be computed in $\mathcal{O}(n)$ time.

Corollary 4.10 (improved version of Theorem 1.11, Corollary 1.14 restated).
Let G be an arbitrary graph on n vertices, $m \in [n]$, and let $(T, \mathcal{X})$ be a tree decomposition of G of width at most $t - 1$. Then G allows an m-cut (B, W) of width at most

$$e_G(B, W) \;\leq\; \min\{\log_2(2m), \log_2(2(n-m))\}\,t\Delta(G) \;\leq\; \log_2(n)t\Delta(G).$$

If $V(G) = [n]$ and the tree decomposition $(T, \mathcal{X})$ is provided as input, a cut with these properties can be computed in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time.

Observe that Corollary 4.10 for $m = \lfloor \frac{1}{2}n \rfloor$ and Theorem 1.11 are very similar. Both consider a graph G on n vertices and a tree decomposition $(T, \mathcal{X})$ of G, which is used to break the graph into smaller pieces in order to construct an m-cut in G for an arbitrary $m \in [n]$ or a bisection in G. The running time of the algorithm in Corollary 4.10 is linear in the size of the input and asymptotically faster than the running time of both algorithms contained in Theorem 1.11.

Furthermore, Theorem 1.11 or, more precisely, its generalized version Theorem 3.8 was used to derive Theorem 3.9, which can now be improved as well. Indeed, consider a planar graph $G = (V, E)$ on n vertices, fix an integer $m \in [n]$, and let $(T, \mathcal{X})$ be a tree decomposition of G of width $t - 1$ with $t \leq \sigma\sqrt{n}$, where $\sigma := \sqrt{8}$. The idea of Theorem 3.9 was to first use separating clusters from $(T, \mathcal{X})$ to break G into smaller pieces and then to use planar separators. The part of the construction using the separating clusters is now replaced by an approximate cut. Assume that $m \leq \frac{1}{2}n$ without loss of generality, as otherwise the black set and the white set can be switched as in the proof of the above corollaries. First, we will show that there is an exact m-cut (B, W) in G with

$$e_G(B, W) \;\leq\; t\left(\left\lceil \log_2\left(\frac{\sigma^2 n}{t^2}\right)\right\rceil + 4 + \sqrt{6}\right)\Delta(G)$$

and then improve the bound to

$$e_G(B, W) \;\leq\; t\left(\left\lceil \log_2\left(\frac{\sigma^2 n}{t^2}\right)\right\rceil + 3 + \sqrt{6}\right)\Delta(G), \tag{4.2}$$

which is the same bound as in Theorem 3.9.

See Figure 4.2 for on overview of the following construction.

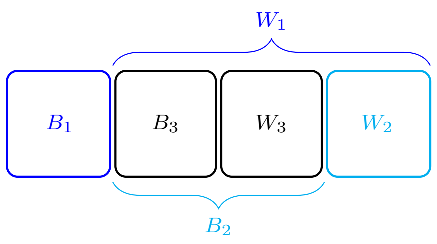


Figure 4.2: Overview of the cuts used in the proof of Corollary 4.11.

Case A: $t = \sigma\sqrt{n}$. Then $\sqrt{n} = \frac{t}{\sigma}$ and Theorem 3.5 implies that G admits an m-cut (B, W), which satisfies $e_G(B, W) \leq t(3 + \sqrt{6})\Delta(G)$ as desired.

Case B: $\sigma\sqrt{2m} < t < \sigma\sqrt{n}$. Define $B_1 := \emptyset$ and $W_1 := V$. Then, $e_G(B_1, W_1) = 0 < \left\lceil \log_2\left(\frac{\sigma^2 n}{t^2}\right)\right\rceil t\Delta(G)$ and $m_3 := m - |B_1| = m < \frac{t^2}{2\sigma^2}$.

Case C: $t \leq \sigma\sqrt{2m}$. Define $c := 1 - \frac{t^2}{2\sigma^2 m}$, which satisfies $c \in [0, 1)$, and let (B_1, W_1) be a c-approximate m-cut in G as in Lemma 4.8b). Then

$$e_G(B_1, W_1) \;\leq\; \left\lceil \log_2\left(\frac{1}{1-c}\right)\right\rceil t\Delta(G) \;=\; \left\lceil \log_2\left(\frac{2\sigma^2 m}{t^2}\right)\right\rceil t\Delta(G) \;\leq\; \left\lceil \log_2\left(\frac{\sigma^2 n}{t^2}\right)\right\rceil t\Delta(G)$$

and

$$m_3 \;:=\; m - |B_1| \;\leq\; m - cm \;=\; \frac{t^2}{2\sigma^2}.$$

So in Case B and Case C

$$m_3 \;\leq\; \frac{t^2}{2\sigma^2} \qquad \text{and} \qquad e_G(B_1, W_1) \;\leq\; \left\lceil \log_2\left(\frac{\sigma^2 n}{t^2}\right)\right\rceil t\Delta(G).$$

The aim is to cut off m_3 vertices from W_1, which can be done with Theorem 3.5. Before doing so, it is ensured via a simple approximate cut that the graph to which Theorem 3.5 is applied does not have too many vertices.

Case 1: $|B_1| = m$. Defining $B := B_1$ gives an m-cut in G of the desired width.

Case 2: $|B_1| < m$. Let $m_2 := 2m_3$, which satisfies $m_2 \geq 1$. Furthermore, W_1 contains enough vertices to cut off m_3 vertices from W_1 as $|W_1| = n - |B_1| \geq m - |B_1| = m_3$.

Case 2a: $m_2 > |W_1|$. Let $B_2 := W_1$ and $W_2 := \emptyset$, which defines a cut in $G[W_1]$ that cuts no edges, i. e., $e_{G[W_1]}(B_2, W_2) = 0$. Moreover, $m_3 \leq |B_2| \leq 2m_3$.

Case 2b: $m_2 \leq |W_1|$. Let (B_2, W_2) be a simple approximate m_2-cut in $G[W_1]$ as obtained by applying Lemma 4.6 with the tree decomposition induced by $G[W_1]$ in $(T, \mathcal{X})$. So $e_{G[W_1]}(B_2, W_2) \leq t\Delta(G)$ and $m_3 \leq |B_2| \leq 2m_3$.

In both cases, Case 2a and Case 2b, $e_{G[W_1]}(B_2, W_2) \leq t\Delta(G)$ and $0 < m_3 \leq |B_2| \leq 2m_3$. So $G[B_2]$ is a planar graph on at most $2m_3 \leq \frac{t^2}{\sigma^2}$ vertices from which we can cut off m_3 vertices. Theorem 3.5 implies that there is an exact m_3-cut (B_3, W_3) in $G[B_2]$ with

$$e_{G[B_2]}(B_3, W_3) \;\leq\; \sigma(3 + \sqrt{6})\Delta(G)\sqrt{\tfrac{t^2}{\sigma^2}} \;=\; (3 + \sqrt{6})\Delta(G)t.$$

Now, defining $B := B_1 \cup B_3$ and $W := V \setminus B$ gives an exact m-cut in G of width

$$\begin{aligned} e_G(B, W) &\leq e_G(B_1, W_1) + e_{G[W_1]}(B_2, W_2) + e_{G[B_2]}(B_3, W_3) \\ &\leq t\left(\left\lceil \log_2\left(\frac{\sigma^2 n}{t^2}\right)\right\rceil + 1 + 3 + \sqrt{6}\right)\Delta(G). \end{aligned} \tag{4.3}$$

Next, the analysis is tightened to achieve the bound in (4.2), by analyzing the c-approximate m-cut (B_1, W_1) in Case C, that was obtained with Algorithm 4.2, more closely. It suffices to look at Case C, as Case A directly derives the tighter bound and the estimate on $e_G(B_1, W_1)$ in Case B is loose. So consider Case C. Clearly, if Algorithm 4.2 executes the outer while loop at most $\left\lceil \log_2\left(\frac{1}{1-c}\right)\right\rceil - 1$ times, then the bound in (4.3) improves in the desired way. It is now argued that, if the outer while loop in Algorithm 4.2 is executed exactly $\left\lceil \log_2\left(\frac{1}{1-c}\right)\right\rceil$ times, then the simple approximate cut (B_2, W_2) can be avoided, thus improving the bound on $e_G(B, W)$ as desired. So assume that Algorithm 4.2 is applied as described above in Case C and the outer while loop is executed exactly $\left\lceil \log_2\left(\frac{1}{1-c}\right)\right\rceil$ times. Denote by B' the state of the set B in Algorithm 4.2 before entering the last execution of the outer while loop. Then, invariant (iii) in the proof of Lemma 4.8 implies that

$$m - |B'| \;\leq\; \frac{1}{2^{\lceil \log_2(\frac{1}{1-c})\rceil - 1}} m \;\leq\; \frac{2}{2^{\log_2(\frac{1}{1-c})}} m \;=\; 2(1-c)m \;=\; \frac{t^2}{\sigma^2} \tag{4.4}$$

by the definition of c in Case C. As argued in the proof of Lemma 4.8, if Lines 16-17 are executed, then the returned set B_1 satisfies $|B_1| = m$ and no simple approximate cut as in Case 2b is needed. So assume that Lines 16-17 are not executed and let i be the node of T that is considered in the last execution of the outer while loop. Denote by $j_1, \ldots, j_k$ the children of i and note that $k \geq 2$ as well as that $1 \leq \ell < k$ for the value of ℓ determined in Lines 9-13. So i has a child $j_{\ell+1}$ and $|B_1| + |\tilde{Y}^{j_{\ell+1}}| > m$ due to Line 10, i. e., $|\tilde{Y}^{j_{\ell+1}}| \geq m_3$. Furthermore, invariant (ii) in the proof of Lemma 4.8 and (4.4) imply $\tilde{y}_{j_{\ell+1}} < m - |B'| \leq \frac{t^2}{\sigma^2}$. So instead of computing B_2 with a simple approximate cut in $G[W_1]$, choose $B_2 := \tilde{Y}^{j_{\ell+1}}$, which is disjoint from B_1 by invariant (i) in the proof of Lemma 4.8 and Proposition 4.7a), and define $W_2 := W_1 \setminus B_2$. Then all edges cut by (B_2, W_2) in $G[W_1]$ are edges in $E_G(i)$ and counted in the bound on $e_G(B_1, W_1)$. So, $G[B_2]$ is planar and contains at most $\frac{t^2}{\sigma^2}$ vertices. Moreover, we have $|B_2| = |\tilde{Y}^{j_{\ell+1}}| \geq m - |B_1| = m_3$. As in the end of Case 2, Theorem 3.5 implies that there is an exact m_3-cut (B_3, W_3) in $G[B_2]$ of width at most $(3 + \sqrt{6})\Delta(G)t$.

Next, let us quickly consider the running time of the procedure presented above. Assume that $V(G) = [n]$. In Case A, the running time is $\mathcal{O}(n)$ by Theorem 3.5 and there is nothing to do in Case B. In Case C, computing the c-approximate m-cut (B_1, W_1) in G takes time proportional to $\|(T, \mathcal{X})\|$ by Lemma 4.8 and returns a list of the vertices in B_1. It is easy to see that without asymptotically increasing the running time, the algorithm can compute a list of the vertices in $\tilde{Y}^{j_{\ell+1}}$ as well. If $|\tilde{Y}^{j_{\ell+1}}| \leq \frac{t^2}{\sigma^2}$, then the algorithm sets $B_2 = \tilde{Y}^{j_{\ell+1}}$. Otherwise, it follows the construction of the cut (B_2, W_2) described in the first version. There is nothing to do in Case 1 and in Case 2a, so from now on assume that Case 2b applies. Then, a list of the vertices in W_1 can be obtained and a bijection between W_1 and the set $\{1, \ldots, |W_1|\}$ can be set up in $\mathcal{O}(n)$ time each by Proposition 2.20 and Lemma 2.21a). A tree decomposition $(T_1, \mathcal{X}_1)$ of $G[W_1]$ can be obtained in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time according to Proposition 2.31b) by using Lemma 2.21c) to check whether a vertex v is in W_1. Then, the simple approximate cut (B_2, W_2) in $G[W_1]$ can be computed in $\mathcal{O}(\|(T_1, \mathcal{X}_1)\|) = \mathcal{O}(\|(T, \mathcal{X})\|)$ time. The subgraph $G[B_2]$ can be computed in $\mathcal{O}(\|G\|)$ time by Corollary 2.23, which simplifies to $\mathcal{O}(n) = \mathcal{O}(\|(T, \mathcal{X})\|)$ by Corollary 2.9 as G is planar. Then, the computation of the exact m_3-cut in $G[B_2]$ with the algorithm contained in Theorem 3.5 takes $\mathcal{O}(n)$ time. So all in all, a running time of $\mathcal{O}(\|(T, \mathcal{X})\|)$ is obtained and the proof of the next corollary is completed.

Corollary 4.11 (improved version of Theorem 3.9).
Let $\sigma = \sqrt{8}$. For every planar graph G on n vertices, every integer $m \in [n]$, and every tree decomposition $(T, \mathcal{X})$ of G of width at most $t-1$ with $t \leq \sigma\sqrt{n}$, there is an m-cut (B, W) in G that satisfies

$$e_G(B, W) \;\leq\; t\left(\left\lceil \log_2\left(\frac{\sigma^2 n}{t^2}\right)\right\rceil + 3 + \sqrt{6}\right)\Delta(G).$$

If $V(G) = [n]$, an m-cut with these properties can be computed in $\mathcal{O}\left(\|(T, \mathcal{X})\|\right)$ time.

Chapter 5

Trees with Long Paths and Tree-Like Graphs Allowing a Tree Decomposition with a Heavy Path

The aim of this chapter is to prove Theorem 1.1 and Theorem 1.3 as well as their algorithmic and improved versions, Theorem 1.8 and Theorem 1.9, which were introduced in Section 1.2. Recall that Theorem 1.1 relates the minimum bisection width of a tree to its diameter. Consider a tree T on $n \geq 2$ vertices. The idea is to use a longest path in T to construct a bisection of small width in T. Clearly, if T is a path, then a bisection in T that cuts at most one edge can be found easily. Also, we will show in Section 5.1 that, if T contains a path on more than $\frac{1}{2}n$ vertices, then a bisection of width at most two in T can be constructed easily. Section 5.1 slowly introduces the methods used throughout the chapter by studying the case when $\operatorname{diam}(T) > \frac{1}{4}n$. Afterwards, in Section 5.2, Theorem 1.1 is derived and then the bound on the minimum bisection width is improved to obtain the bound in Theorem 1.8 by refining the proof of Theorem 1.1. To complete the proof of Theorem 1.8, a linear-time algorithm that computes a bisection within the improved bound is presented. Section 5.3 generalizes the results to tree-like graphs. Here, only the stronger bound on the minimum bisection width from Theorem 1.9 is derived. Moreover, two algorithms that, when given a tree decomposition, compute a bisection in the underlying graph of width within this bound are discussed. The first one does not run in linear time for all tree decompositions, but the second one, whose implementation is more involved, runs in linear time.

5.1 Getting to Know the Techniques

This section focuses on trees and the special case of finding a bisection of small width in a bounded-degree tree on n vertices whose diameter is at least $\frac{1}{4}n$. The proof of this result uses the same technique as the remaining chapter, but in a simplified form. First, we introduce a new notation using the diameter. Here and in Section 5.2, we will work with longest paths in trees, i. e., paths whose length is the diameter. In order to compare the diameters of two graphs with possibly different numbers of vertices and also to generalize to forests, the following concept is used. If G is a forest on n vertices and $G_1, \ldots, G_\ell$ are the components of G, then define the *relative diameter* of G as

$$\operatorname{diam}^*(G) := \frac{1}{n}\sum_{h=1}^{\ell}(\operatorname{diam}(G_h)+1) = \frac{1}{n}\sum_{h=1}^{\ell}|V(P_h)|,$$

where P_h is a longest path in G_h for every $h \in [\ell]$. Note that, for a tree T, the relative diameter of T denotes the fraction of vertices on a longest path in T. Observe that $\operatorname{diam}^*(G) \le 1$ for every forest G, and $\operatorname{diam}^*(P) = 1$ for every path P. Using this notation, the aim of this section is to prove the following theorem, which can be seen as a forerunner of Theorem 1.1.

Theorem 5.1.
Every forest G with an even number of vertices and $\operatorname{diam}^*(G) > \frac{1}{4}$ allows a bisection (B, W) that satisfies $e_G(B, W) \le 6\Delta(G)$.

Before starting to discuss the proof of Theorem 5.1, two lemmas are presented. The first one is a short technical lemma, that will be useful to generalize many results from trees to forests throughout this chapter.

Lemma 5.2.

a) For every forest G on n vertices with $\Delta(G) \ge 3$, there is a tree T on n vertices such that $G \subseteq T$, $\Delta(T) = \Delta(G)$, and $\operatorname{diam}^*(T) = \operatorname{diam}^*(G)$. Then, every cut (B, W) in T is also a cut in G and satisfies $e_G(B, W) \le e_T(B, W)$.

b) Let $m, n \in \mathbb{N}$ with $m \le n$. Then, every forest G on n vertices with $\Delta(G) \le 2$ satisfies $\operatorname{diam}^*(G) = 1$ and allows an m-cut of width at most $\Delta(G)$.

Proof.

a) Let G be an arbitrary forest on n vertices with $\Delta(G) \ge 3$. If G is connected, choose $T = G$, which satisfies all requirements. So assume that G is not connected. Denote by ℓ the number of components of G and let $G_1, \dots, G_\ell$ be the components of G. For every $h \in [\ell]$, consider a longest path P_h in G_h. Note that $\operatorname{diam}^*(G) = \frac{1}{n}\sum_{h=1}^{\ell}|V(P_h)|$. For $h \in [\ell]$, if P_h consists of at least two vertices, let x_h and y_h be the two leaves of P_h. Otherwise, P_h consists of only one vertex v and we set $x_h = v$ and $y_h = v$. Next, for every $h \in [\ell - 1]$, insert the edge $\{y_h, x_{h+1}\}$ into G and denote by T the tree obtained in this way. The tree T contains a path that uses all the vertices in the path P_h for every $h \in [\ell]$. Therefore, $\operatorname{diam}^*(T) \ge \operatorname{diam}^*(G)$. Furthermore, $\deg_G(x_h) \le 1$ and $\deg_G(y_h) \le 1$ for every $h \in [\ell]$, as otherwise the path P_h is not a longest path in the tree G_h. Hence, $\deg_T(x_h) \le 2$ and $\deg_T(y_h) \le 2$ for all $h \in [\ell]$, which implies that $\Delta(T) = \Delta(G)$. To show that $\operatorname{diam}^*(G) \ge \operatorname{diam}^*(T)$, consider a longest path P in T. Then, $\operatorname{diam}^*(T) = \frac{1}{n}|V(P)|$. Furthermore, by deleting the edges not in G from P a collection of paths in G with at most one path in each component of G is obtained. Therefore, $\operatorname{diam}^*(G) \ge \frac{1}{n}|V(P)| = \operatorname{diam}^*(T)$ and consequently $\operatorname{diam}^*(T) = \operatorname{diam}^*(G)$.

   As the forest G and the tree T have the same vertex set, every cut in T is also a cut in G and satisfies $E_G(B, W) \subseteq E_T(B, W)$ due to $E(G) \subseteq E(T)$.

b) Fix $n \in \mathbb{N}$ and $m \in [n]$. Let $G = (V, E)$ be an arbitrary forest on n vertices with $\Delta(G) \le 2$. As every component of G is a path, it follows that $\operatorname{diam}^*(G) = 1$. To construct the set B of an m-cut in G, greedily collect m vertices by traversing the paths successively. Then, the cut (B, W) with $W := V \setminus B$ is an m-cut of width at most $\Delta(G)$. □

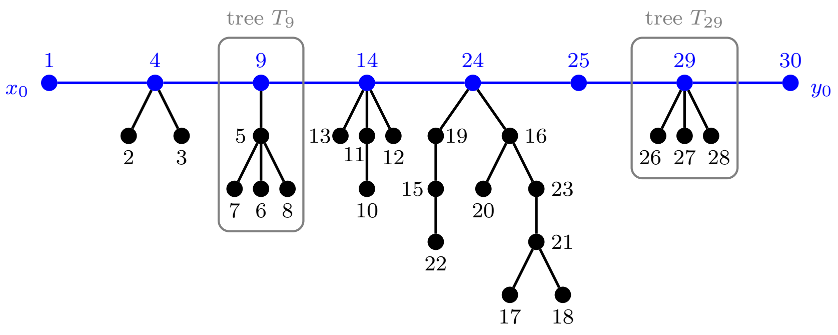


Figure 5.1: Example of a P-labeling in a tree T. The path P is colored blue and drawn on the top.

The second lemma will be a tool later on in the proof of Theorem 5.1 and gives a first impression on how a long path in a tree can be used to find a bisection of small width.

Lemma 5.3.
Let $m, n \in \mathbb{N}$ with $m \leq n$. Then, every forest G on n vertices that satisfies $\operatorname{diam}^*(G) > \frac{1}{2}$, *allows an m-cut (B, W) with $e_G(B, W) \leq 2$.*

Note that the bounds on the width of the cut in Theorem 1.1 and Theorem 5.1 both depend on the maximum degree of the considered graph, but the bound on the width of the cut in the previous lemma does not depend on the maximum degree. The proof of the previous lemma uses a labeling of the vertices that roughly follows along a path in each component of the forest. More precisely, consider a tree $T = (V, E)$ on n vertices and a path $P = (V_P, E_P)$ in T. Let x_0 and y_0 be the ends of P. For every $v \in V_P$, denote by T_v the component of $T - E_P$ that contains the vertex v. The vertices of some subgraph $H \subseteq T$ are said to have *consecutive labels* if there are two integers $0 \leq \ell < \ell'$ such that every vertex of H has a label in $\{\ell, \ell+1, \ldots, \ell'\}$ and vice versa. A labeling of the vertices of T with $1, 2, \ldots, n$ is called a *P-labeling* if

- for each $v \in V_P$, the vertices of T_v have consecutive labels and v receives the largest label among those, and
- for all $v, v' \in V_P$ with $v \neq v'$, if x_0 is closer to v than to v', then the label of v is smaller than the label of v'.

Figure 5.1 shows an example of a P-labeling. In this figure and the following figures that depict a situation involving a P-labeling of a tree, the path P will be drawn on the top and the trees T_v for $v \in V(P)$ will be hanging down from this path. Sometimes, the trees T_v with $v \in V_P$ are not drawn explicitly and are only indicated by triangles. Clearly, for every tree T and every path $P \subseteq T$, such a P-labeling exists. Furthermore, when considering a P-labeling of a tree T, the *path-vertex* of a vertex $x \in V(T)$ is defined to be the unique vertex $v \in V(P)$ with $x \in V(T_v)$.

Proof of Lemma 5.3. Let $G = (V, E)$ be an arbitrary forest that satisfies $\operatorname{diam}^*(G) > \frac{1}{2}$ and denote by n the number of vertices of G. Fix an $m \in [n]$. First, note that due to Lemma 5.2 it suffices to consider the case when G is a tree. Denote by $d := \operatorname{diam}^*(G)$ the relative diameter of G and let $P = (V_P, E_P)$ be a longest path in G. Let x_0 and y_0 be the ends of P, and note that $|V_P| = dn$. Consider a P-labeling of the vertices of T and identify each vertex with its label. From now on, a number that differs by a multiple of n from a label is considered to be the same as this label. Define $N_m(v) := v + m$ and note that $N_m : V \to V$ is a bijection. Then, as $|V_P| = dn > \frac{1}{2}n$, there must be a vertex $v \in V_P$

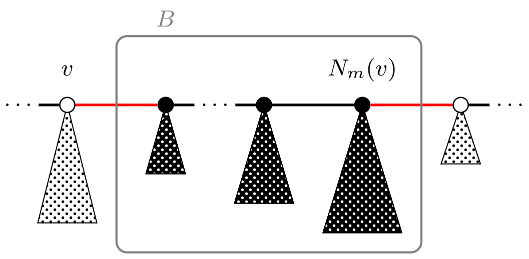


Figure 5.2: Proof of Lemma 5.3. Construction of the cut (B, W).

with $N_m(v) \in V_P$. Define $B := \{v+1, v+2, \dots, v+m\}$ and $W := V \setminus B$. The cut (B, W) cuts at most two edges in G. If $m \neq n$, these are the edge connecting v with its neighbor in $B \cap V_P$ and the edge connecting $N_m(v)$ with its neighbor in $W \cap V_P$, see Figure 5.2. Note that one of these edges might not exist, for example if $N_m(v) = y_0$. □

Proof of Theorem 5.1. Let $G = (V, E)$ be a forest on n vertices with $\operatorname{diam}^*(G) > \frac{1}{4}$. Assume that n is even. Due to Lemma 5.2 it suffices to consider the case when G is a tree with $\Delta(G) \geq 3$. Let $P = (V_P, E_P)$ be a longest path in G and denote by x_0 and y_0 the ends of P. Furthermore, define $d := \operatorname{diam}^*(G)$ and note that $|V_P| = dn$. Moreover, for $v \in V_P$, let T_v be the component of $T - E_P$ that contains the vertex v and define $T'_v := V(T_v) \setminus \{v\}$. Consider a P-labeling of the vertices of G, where x_0 received label 1, and identify each vertex with its label. From now on, a number that differs by a multiple of n from a label is considered to be the same as this label. Denote by $A(v) := v + \frac{1}{2}n$ the *antipole* of v for each vertex $v \in V$ and note that $A : V \to V$ is a bijection.

Case 1: There is a vertex $v \in V_P$ with $A(v) \in V_P$. Then choosing $B := \left\{v+1, \dots, v+\frac{1}{2}n\right\}$ and $W := V \setminus B$ gives a bisection of width at most 2, which is similar to the cut used in the proof of Lemma 5.3.

Case 2: $A(v) \notin V_P$ for all $v \in V_P$. In this case, finding the cut is more involved and requires some new notation. For sets $U \subseteq V$ let $A(U) := \{A(u) \colon u \in U\}$. As $\Delta(G) \geq 3$ it follows that $x_0 \neq y_0$ and the graph G does not contain the edge $\{x_0, y_0\}$. Denote by G^+ the graph obtained from G by adding the edge $\{x_0, y_0\}$. The following definitions are visualized in Figure 5.3. A vertex $v \in V_P$ is called *special* if there is a vertex $x \in T'_v$ with $A(x) \in V_P$. For every special vertex $v \in V_P$, define $P_v := A(T'_v) \cap V_P$. Note that P_v induces a path or a cycle in G^+, and that $\{P_v \colon v \in V_P \text{ is special}\}$ is a partition of V_P. Also, for every special vertex $v \in V_P$, let $v^P := A(y)$ for the smallest vertex $y \in T'_v$ with $A(y) \in V_P$. Then, for every special $v \in V_P$, we have $v^P \in P_v$ and v^P is an end of the path induced by P_v in G^+, if P_v does not induce a cycle in G^+. Moreover, for every special $v \in V_P$, let H_v be the union of the sets T'_x for all $x \in P_v$ with $x \neq v^P$.

After some accounting, we will pick a special vertex $v \in V_P$ and use its sets P_v and H_v to construct a bisection in G. By construction, every special $v \in V_P$ satisfies

$$|T'_v| \geq |P_v| + |H_v|. \tag{5.1}$$

Recall that $|V_P| = dn > \frac{1}{4}n$ and that $\{P_v \colon v \in V_P \text{ is special}\}$ is a partition of V_P. Therefore,

$$\frac{1}{4}n \;<\; |V_P| \;=\; \sum_{\substack{v \in V_P: \\ v \text{ is special}}} |P_v| \qquad \text{and} \tag{5.2}$$

$$\frac{3}{4}n \;>\; |V \setminus V_P| \;=\; \sum_{\substack{v \in V_P: \\ v \text{ is special}}} \left(|H_v| + |T'_{v^P}|\right). \tag{5.3}$$

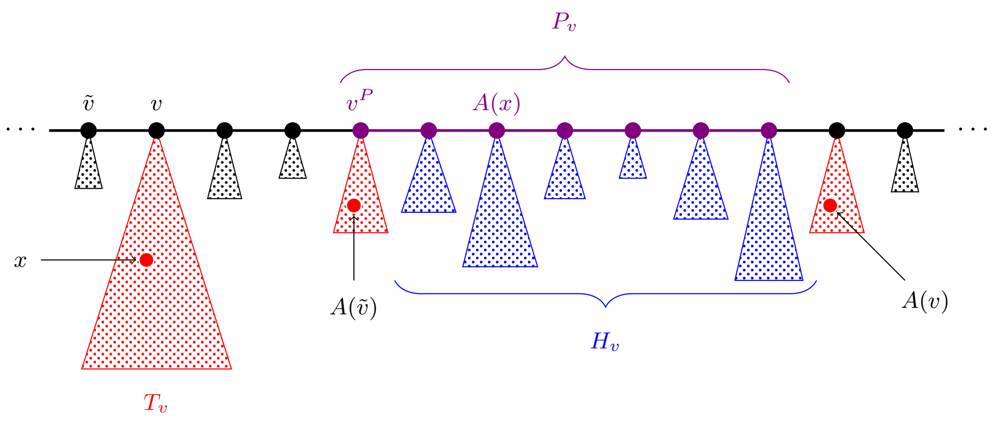


Figure 5.3: Proof of Theorem 5.1, notation in Case 2. A tree T_v is colored red if it contains a vertex w with $A(w) \in V_P$, i. e., if the vertex v is special.

Claim 5.4.
For every special vertex $v \in V_P$, the vertex $w = v^P$ is special.

Indeed, consider a special vertex $v \in V_P$ and denote by $\tilde{v}$ the unique vertex in $V_P \setminus \{v\}$ with $\tilde{v} + 1 \in T'_v$, i. e., $\tilde{v}$ is the vertex obtained from v by walking one step along the unique cycle of G^+ against the numeration. Due to the assumption of Case 2, the vertex $A(\tilde{v})$ is not in V_P and, hence, $A(\tilde{v})$ must be in T'_{v^P}, see Figure 5.3. Now, $A(A(\tilde{v})) = \left(\tilde{v} + \frac{1}{2}n\right) + \frac{1}{2}n = \tilde{v} \in V_P$ and, therefore, v^P is special. This completes the proof of Claim 5.4.

Claim 5.5.
A vertex $w \in V_P$ is special if and only if there is a special vertex $v \in V_P$ with $v^P = w$.

Indeed, by Claim 5.4 it suffices to show that there are as many special vertices $w \in V_P$ as there are vertices v^P for a special vertex $v \in V_P$. To do so, note that for two distinct special vertices v and w, the vertices v^P and w^P are distinct. Hence,

$$\left|\left\{v \in V_P \colon\ v \text{ is special}\right\}\right| \ \leq\ \left|\left\{v^P \colon\ v \in V_P \text{ is special}\right\}\right| \overset{\text{Claim 5.4}}{\leq} \left|\left\{w \in V_P \colon\ w \text{ is special}\right\}\right|.$$

Observing that all these inequalities are equalities completes the proof of Claim 5.5.

Now, Claim 5.5 implies that

$$\sum_{\substack{w \in V_P\colon \\ w \text{ is special}}} |T'_w| \ = \sum_{\substack{v \in V_P\colon \\ v \text{ is special}}} |T'_{v^P}|.$$

The previous equation can be used to rewrite (5.3), which gives

$$\frac{3}{4}\,n \ > \sum_{\substack{v \in V_P\colon \\ v \text{ is special}}} \left(|H_v| + |T'_v|\right).$$

With (5.2) it follows that

$$3 \sum_{\substack{v \in V_P\colon \\ v \text{ is special}}} |P_v| \ > \ \frac{3}{4}\,n \ > \sum_{\substack{v \in V_P\colon \\ v \text{ is special}}} \left(|H_v| + |T'_v|\right).$$

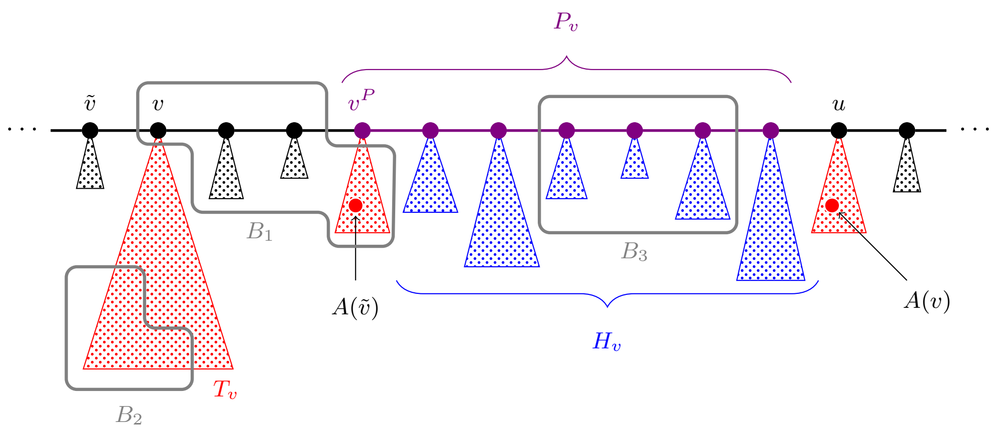


Figure 5.4: Proof of Theorem 5.1. Construction of the bisection (B, W) with $B = B_1 \dot\cup B_2 \dot\cup B_3$ in Case 2.

Consequently, there is a special vertex $v \in V_P$ with

$$|T'_v| + |H_v| \ < \ 3\,|P_v|. \tag{5.4}$$

Replacing $|T'_v|$ with (5.1) yields

$$|P_v| + 2\,|H_v| \ < \ 3\,|P_v| \qquad \Rightarrow \quad 2\,|P_v| + 2\,|H_v| \ < \ 4\,|P_v| \qquad \Rightarrow \quad |P_v| + |H_v| < 2\,|P_v|.$$

Therefore, the set $Z := P_v \dot\cup H_v$ satisfies $|Z| < 2|P_v|$. Since v is special, the set P_v is nonempty and, thus, $Z \neq \emptyset$. The vertices in P_v induce a path or a cycle in G^+ and a collection of paths in $G[Z]$ with at most one path in each component of $G[Z]$. This implies that $\operatorname{diam}^*(G[Z]) \geq \frac{|P_v|}{|Z|} > \frac{1}{2}$.

Next, the vertex v and the set Z are used to construct the set B for the bisection from three disjoint parts $B_1 \subseteq V \setminus (T'_v \cup Z)$, $B_2 \subseteq T'_v$, and $B_3 \subseteq Z$. See also Figure 5.4 for a visualization of the following definitions. If $v = v^P$, let $B_1 := \emptyset$ and otherwise let $B_1 := \{v, v+1, \ldots, v^P - 1\}$. Then, $|B_1| < \frac{1}{2}n$ as $v^P - \frac{1}{2}n$ is in T'_v, i. e., the vertex that is mapped to v^P by A is in T'_v. Therefore, $m_2 := \frac{1}{2}n - |B_1|$ satisfies $1 \leq m_2 \leq |T'_v|$. We would like to cut off m_2 vertices from $G[Z]$ but Z might not contain enough vertices to do so. Therefore, we first cut off some vertices from $G[T'_v]$. Let (B_2, W_2) be a $\frac{2}{3}$-approximate m_2-cut in $G[T'_v]$ with $e_{G[T'_v]}(B_2, W_2) \leq 2\Delta(G)$, which exists according to Lemma 4.5b). Recall that this implies that $\frac{2}{3}m_2 \leq |B_2| \leq m_2$. Let $m_3 := \frac{1}{2}n - |B_1| - |B_2|$ and note that B_1 and B_2 are disjoint. Then,

$$0 \ \leq \ m_3 \ = \ \left(\tfrac{1}{2}n - |B_1|\right) - |B_2| \ \leq \ m_2 - \tfrac{2}{3}m_2 \ \leq \ \tfrac{1}{3}|T'_v| \ \overset{(5.4)}{<} \ |P_v| \ \leq \ |Z|.$$

Therefore, the graph $G[Z]$ contains at least m_3 vertices. If $m_3 = 0$, define $B_3 := \emptyset$. Otherwise, Lemma 5.3 guarantees that there is an m_3-cut (B_3, W_3) in $G[Z]$ that satisfies $e_{G[Z]}(B_3, W_3) \leq 2 \leq \Delta(G)$. Now, (B, W) with $B := B_1 \dot\cup B_2 \dot\cup B_3$ and $W := V \setminus B$ is a bisection in G.

Next, the number of edges cut by (B, W) in G is estimated. Let $\tilde{V} := V \setminus (T'_v \cup B_1 \cup Z)$ and denote by $u \in V_P$ the vertex with $A(v) \in T'_u$. Then, the cut $(T'_v, B_1, Z, \tilde{V})$ in G cuts only edges incident to v, v^P, and u, i. e., at most $3\Delta(G)$ edges. Together with the bounds on the number of cut edges of the cut (B_2, W_2) in $G[T'_v]$ and the cut (B_3, W_3) in $G[Z]$, it follows that

$$\begin{aligned} e_G(B, W) \ &\leq \ e_G(T'_v, B_1, Z, \tilde{V}) + e_{G[T'_v]}(B_2, W_2) + e_{G[Z]}(B_3, W_3) \\ &\leq \ 3\Delta(G) + 2\Delta(G) + \Delta(G) \\ &\leq \ 6\Delta(G). \end{aligned}$$

□

Recall that, in Section 1.2.6, it was claimed that Theorem 1.1 can be generalized to m-cuts. So there needs to be also a generalization of Theorem 5.1 to m-cuts. However, the above proof works only for bisections and only if the number of vertices of the considered forest is even. The reason for this is that we used $A(A(v)) = v$ when proving Claim 5.4, which does not hold anymore when $A(v)$ is replaced by $N_m(v) = v + m$ as used in the proof of Lemma 5.3 to find an m-cut. In order to deal with this, we will consider a backward and a forward version of being special and also define a backward and a forward version of the sets P_v and H_v. The details are omitted as the purpose of Theorem 5.1 was to get to know special vertices and the sets P_v and H_v. In Section 5.2.2, these more general versions are used to prove a lemma, which is the heart of the proof of Theorem 1.1.

To conclude this section, we discuss how the ideas presented in the previous theorem can be generalized to prove the bound in Theorem 1.1. Although the minimum bisection problem asks for a partition of the vertex set into two sets of (almost) the same size, we need to take a more general approach and consider m-cuts in order to apply induction. The main idea of the induction is to double the relative diameter in each round: If, for the desired value of m, an m-cut (B, W) in the forest G with $e_G(B, W) \leq 2$ cannot be found easily, more precisely, if an analogon of Case 1 from the proof of Theorem 5.1 does not apply, then we construct a subgraph $G[Z]$ with $\operatorname{diam}^*(G[Z]) \geq 2 \operatorname{diam}^*(G)$. This idea appears also in the proof of Theorem 5.1, where a subgraph $G[Z]$ with $\operatorname{diam}^*(G[Z]) > \frac{1}{2}$ was constructed if Case 1 did not apply. Then, Lemma 5.3 was used to find an m'-cut in $G[Z]$. Here, it is necessary that Lemma 5.3 is for m-cuts and not only bisections. In general, Lemma 5.3 cannot be applied to $G[Z]$, but the same method can be applied iteratively to $G[Z]$ until a graph with relative diameter greater than $\frac{1}{2}$ is obtained. In Section 5.2.1, Theorem 1.1 is proved by an induction that doubles the relative diameter of the considered forest in each round.

5.2 Results for Trees

In this section, the proofs for Theorem 1.1 and Theorem 1.8 are presented. Both theorems state an upper bound on the minimum bisection width in a tree in terms of its diameter. Theorem 1.8 additionally claims that a bisection within this bound can be computed in linear time. Sections 5.2.1-5.2.3 focus on constructing a bisection to prove these bounds. First, in Section 5.2.1, the bound in Theorem 1.1 is derived from a doubling-lemma, which hides most of the technical details and whose proof is presented in Section 5.2.2. In both subsections, everything is kept as simple as possible and, therefore, Section 5.2.3 discusses how to tighten the analysis in order to obtain the stronger bound presented in Theorem 1.8. All algorithmic aspects are discussed in Section 5.2.4.

5.2.1 Upper Bound for the Width of Exact Cuts in Trees

The aim of this subsection is to prove the following theorem for exact cuts in forests. As the next theorem is for forests, it uses the relative diameter instead of the diameter, which is used in Theorem 1.1. Observing that every tree T on n vertices satisfies

$$\frac{1}{\operatorname{diam}^*(T)} = \frac{n}{\operatorname{diam}(T) + 1} \leq \frac{n}{\operatorname{diam}(T)}$$

shows that the next theorem implies Theorem 1.1.

Theorem 5.6 (Theorem 1.1 restated and generalized).
For all $n \in \mathbb{N}$ and all $m \in [n]$, every forest G allows an m-cut (B, W) with $e_G(B, W) \leq \dfrac{8\Delta(G)}{\operatorname{diam}^*(G)}$.

The heart of the proof of Theorem 1.1 is the following doubling lemma, that can be used for an induction.

Lemma 5.7.
For all forests G on n vertices and for all $m \in [n]$, there is a cut (B, W, Z) in G such that one of the following two options is satisfied:

1) $Z = \emptyset$, $|B| = m$, *and* $e_G(B, W, Z) \leq 2$, *or*
2) $Z \neq \emptyset$, $|Z| \leq \frac{1}{2}n$, $|B| \leq m \leq |B| + |Z|$, $e_G(B, W, Z) \leq \dfrac{2\Delta(G)}{\operatorname{diam}^*(G)}$, *and* $\operatorname{diam}^*(G[Z]) \geq 2\operatorname{diam}^*(G)$.

Consider a forest G on n vertices and fix an $m \in [n]$. The previous lemma says that there is a cut (B, W) where B has the desired size and few edges are cut, or there is a cut with an additional set Z such that the set B is smaller and the set $B \,\dot\cup\, Z$ is larger than the required size m. In the latter case, the set Z has the additional feature that the relative diameter of $G[Z]$ is at least twice as large as that of G. Applying Lemma 5.7 iteratively to the graph $G' := G[Z]$ with size-parameter $m' := m - |B|$, the relative diameter is doubled in each round until Option 1) occurs and completes the proof or the relative diameter exceeds $\frac{1}{2}$ and the proof can be completed with Lemma 5.3. Note that, as the relative diameter of G increases, the bound on the number of edges cut in Option 2) decreases. Moreover, observe that Lemma 5.7 implies Lemma 5.3 as Option 2) is infeasible when the considered forest G satisfies $\operatorname{diam}^*(G) > \frac{1}{2}$ as $\operatorname{diam}^*(H) \leq 1$ for every forest H.

Next, Lemma 5.7 is used to derive Theorem 5.6, before the proof of Lemma 5.7 is presented in Section 5.2.2.

Proof of Theorem 5.6. The idea is to prove the following claim by induction.

Claim 5.8.
For every $d \in \mathbb{N}$, for every forest G on n vertices with $\operatorname{diam}^*(G) > \frac{1}{2^d}$, and for every $m \in [n]$, there is an m-cut (B, W) in G with $e_G(B, W) \leq 4 \cdot 2^d \Delta(G)$.

To see that the claim implies Theorem 5.6, consider an arbitrary forest G on n vertices and fix some $m \in [n]$. Choose $d^* \in \mathbb{N}$ with $\frac{1}{2^{d^*-1}} \geq \operatorname{diam}^*(G) > \frac{1}{2^{d^*}}$. Then, Claim 5.8 implies that there is an m-cut (B, W) in G with

$$e_G(B, W) \;\leq\; 4 \cdot 2^{d^*} \Delta(G) \;\leq\; \frac{8\Delta(G)}{\operatorname{diam}^*(G)},$$

as desired.

Next, Claim 5.8 is proved by induction over d.

Base ($d = 1$): Let G be an arbitrary forest on n vertices with $\operatorname{diam}^*(G) > \frac{1}{2}$ and fix an arbitrary $m \in [n]$. If $\Delta(G) \leq 2$ then Lemma 5.2b) suffices. Otherwise $\Delta(G) \geq 3$ and Lemma 5.3 shows that the desired m-cut exists.

Step: Fix a $d \geq 2$ and consider an arbitrary forest G with $\operatorname{diam}^*(G) > \frac{1}{2^d}$. Let n be the number of vertices of G and fix some $m \in [n]$. Again, if $\Delta(G) \leq 2$, then Lemma 5.2b) suffices. So, from now on, assume that $\Delta(G) \geq 3$. Apply Lemma 5.7 to G with size-parameter m to obtain a cut $(\tilde{B}, \tilde{W}, \tilde{Z})$ in G.

Case 1: $|\tilde{B}| = m$.
Defining $B := \tilde{B}$ and $W := V \setminus B$ gives the desired cut since

$$e_G(B, W) \;\leq\; \max\{2,\, 2 \cdot 2^d \Delta(G)\} \;=\; 2 \cdot 2^d \Delta(G).$$

Case 2: $|\tilde{B}| < m \leq |\tilde{B}| + |\tilde{Z}|$.

In this case

$$e_G(\tilde{B}, \tilde{W}, \tilde{Z}) \;\leq\; \frac{2\Delta(G)}{\operatorname{diam}^*(G)} \;<\; 2 \cdot 2^d \Delta(G)$$

and $\operatorname{diam}^*(G[\tilde{Z}]) \geq 2 \operatorname{diam}^*(G) > \frac{1}{2^{d-1}}$. Let $G' := G[\tilde{Z}]$ and $n' := |\tilde{Z}|$ be the number of vertices of G'. Set $m' := m - |\tilde{B}|$, which satisfies $m' \leq |\tilde{Z}| = n'$. Now, the induction hypothesis implies that there is an m'-cut (B', W') in G' with $e_{G'}(B', W') \leq 4 \cdot 2^{d-1}\Delta(G)$. Defining $B := \tilde{B} \,\dot\cup\, B'$ and $W := V(G) \setminus B$ produces an m-cut (B, W) in G with

$$\begin{aligned} e_G(B, W) \;&\leq\; e_G(\tilde{B}, \tilde{W}, \tilde{Z}) + e_{G'}(B', W') \\ &\leq\; 2 \cdot 2^d \Delta(G) + 4 \cdot 2^{d-1}\Delta(G) \;=\; 4 \cdot 2^d \Delta(G). \end{aligned}$$

□

5.2.2 Proof of the Doubling Lemma for Trees

This subsection concerns the proof of Lemma 5.7. The proof uses the same ideas as the proof of Theorem 5.1 to find the cut (B, W) or the cut (B, W, Z) with the additional set Z. Again, a P-labeling of the vertices for a longest path P will be used and some vertices in P will be called special. However, since the aim is to find an m-cut and not a bisection, the bijection $N_m(v) = v + m$, which is defined as in the proof of Lemma 5.3, is used instead of the bijection $A(v) = v + \frac{1}{2}n$. As discussed after the proof of Theorem 5.1, some ideas used in the proof of Theorem 5.1 do not work for the bijection $N_m(v)$ for general m. To generalize these ideas, a backward and a forward version of special vertices is defined, which also leads to backward and forward versions of the sets P_v and H_v. Table 5.1 provides an overview on the notation used in the proof of Theorem 5.1 and Lemma 5.7.

Let $G = (V, E)$ be an arbitrary forest on n vertices and fix some $m \in [n]$. If $\Delta(G) \leq 2$, then Lemma 5.2b) implies that a cut (B, W) satisfying Option 1) exists. So assume that $\Delta(G) \geq 3$. In this case, Lemma 5.2a) implies that we may assume that G is a tree. Set $d := \operatorname{diam}^*(G)$ and let $P = (V_P, E_P)$ be a longest path in G. Note that $|V_P| = dn$. As $\Delta(G) \geq 3$, the path P consists of at least three vertices and has two distinct leaves, which are denoted by x_0 and y_0. Let G^+ be the graph obtained from G by inserting the edge $\{x_0, y_0\}$ and note that the vertices in V_P induce a cycle in G^+. For technical reasons, we will sometimes refer to the pair $\{x_0, y_0\}$ as an edge of G, even though it is not. For each vertex $v \in V_P$, let T_v be the component of $G - E_P$ that contains v and define $T'_v := V(T_v) \setminus \{v\}$. Consider a P-labeling of the vertices of G, where x_0 receives label 1, and identify each vertex with its label. From now on, any number that differs from a label in $[n]$ by a multiple of n is considered to be the same as this label. For three vertices $a, b, c \in V$ with $a \neq c$, we say that b *is between* a *and* c if $b = a$, $b = c$, or if starting at a and going along the numeration given by the labeling reaches b before c. If $a = c$, then we say that b is between a and c if $b = a = c$. For example, when $n = 10$, then 5 is between 1 and 7, and 9 is between 8 and 3. For a vertex $v \in V_P$, the unique vertex $w \in V_P$ with the property that T_w contains the vertex $v + 1$ is called the *vertex after* v *on* P. Furthermore, if w is the vertex after v on P, then the edge $\{v, w\}$ is referred to as the *edge after* v *on* P. Similarly, in this case, the edge $\{v, w\}$ is called the *edge before* w *on* P and v is called the *vertex before* w *on* P. For each vertex $v \in V$, the vertex $N_m(v) := v + m$ is called the *m^{th}-next vertex of* v. Note that $N_m : V \to V$ is a bijection and, hence, its inverse function N_m^{-1} is well-defined. For a set $U \subseteq V$, define $N_m(U) := \{N_m(u) \colon u \in U\}$ and $N_m^{-1}(U) := \left\{N_m^{-1}(u) \colon u \in U\right\}$.

Case 1: There is a vertex $v \in V_P$ with $N_m(v) \in V_P$.

Let $v \in V_P$ be a vertex with $N_m(v) \in V_P$ and let B be the set of vertices between $v + 1$ and $N_m(v)$, which satisfies $|B| = m$. Moreover, let $W := V \setminus B$ and $Z := \emptyset$. The cut (B, W) cuts at most two edges. More

Common notation in the proof of Theorem 5.1 and the proof of Lemma 5.7
$G = (V, E)$ forest on n vertices, may assume that G is a tree, $d := \operatorname{diam}^*(G)$, $P = (V_P, E_P)$ longest path in G, ends x_0 and y_0 $\quad\Rightarrow \lvert V_P\rvert = dn$, consider a P-labeling of the vertices, identify each vertex with its label, T_v = components of $G - E_P$, $\;T'_v = V(T_v) \setminus \{v\}$.

Proof of Theorem 5.1	Proof of Lemma 5.7		
only for $d > \frac{1}{4}$ and n even, aim: bisection.	no restrictions on d and n, aim: m-cut.		
	backward version:		forward version:
$A(v) := v + \frac{1}{2}n$,	$N_m^{-1}(v) = v - m$,		$N_m(v) = v + m$,
$v \in V_P$ special if $\exists\, x \in T'_v$ with $A(x) \in V_P$,	$v \in V_P$ b-special if $\exists\, x \in T'_v$ with $N_m^{-1}(x) \in V_P$,		$v \in V_P$ f-special if $\exists\, x \in T'_v$ with $N_m(x) \in V_P$,
$P_v = A(T'_v) \cap V_P$, $\big\{P_v \colon\ v \in V_P \text{ special}\big\}$ is a partition of V_P,	$P_v^b = N_m^{-1}(T'_v) \cap V_P$, $\big\{P_v^b \colon\ v \in V_P \text{ b-special}\big\}$ is a partition of V_P,		$P_v^f = N_m(T'_v) \cap V_P$, $\big\{P_v^f \colon\ v \in V_P \text{ f-special}\big\}$ is a partition of V_P,
$v^P = A(x)$ for the smallest $x \in T'_v$ with $A(x) \in V_P$,	$v^b = N_m^{-1}(x)$ for the smallest $x \in T'_v$ with $N_m^{-1}(x) \in V_P$,		$v^f = N_m(x)$ for the smallest $x \in T'_v$ with $N_m(x) \in V_P$,
$H_v = \bigcup_{x \in P_v \setminus \{v^P\}} T'_x$,	$H_v^b = \bigcup_{x \in P_v^b \setminus \{v^b\}} T'_x$,		$H_v^f = \bigcup_{x \in P_v^f \setminus \{v^f\}} T'_x$,
$\exists$ special $v \in V_P$ with $\lvert T'_v\rvert + \lvert H_v\rvert < 3\lvert P_v\rvert$,	$\exists$ b-special $v \in V_P$ with $\lvert T'_v\rvert + \lvert H_v^b\rvert \le \left(\frac{1}{d} - 1\right) \lvert P_v^b\rvert$,	or	f-special $v \in V_P$ with $\lvert T'_v\rvert + \lvert H_v^f\rvert \le \left(\frac{1}{d} - 1\right) \lvert P_v^f\rvert$,
$Z = P_v \,\dot\cup\, H_v$.	$Z = P_v^b \,\dot\cup\, H_v^b$.	or	$Z = P_v^f \,\dot\cup\, H_v^f$.

Table 5.1: Overview of the notation used in the proofs of Theorem 5.1 and Lemma 5.7.

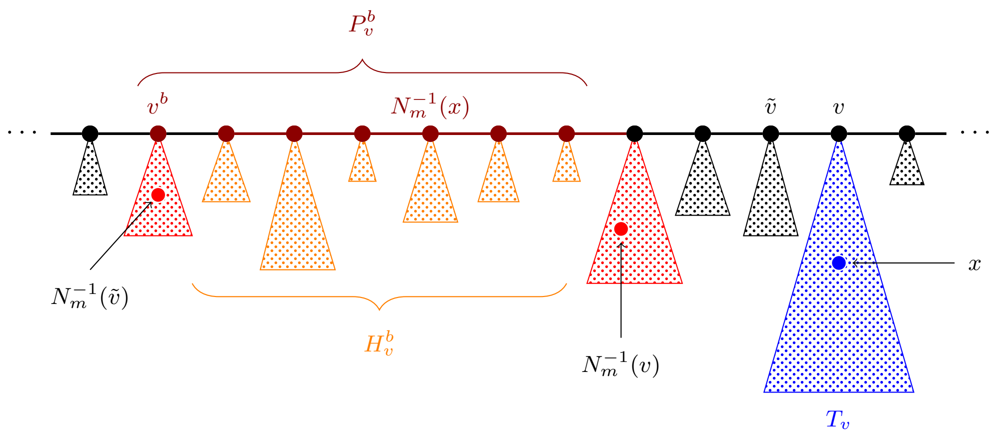
precisely, if they exist, the cut (B, W) cuts the edge after v on P and the edge after $N_m(v)$ on P, except when $m = n$ and no edge is cut. See also Figure 5.2, which refers to Lemma 5.3 where the same situation arose. Hence, the cut (B, W, Z) satisfies Option 1).

Case 2: There is no vertex $v \in V_P$ with $N_m(v) \in V_P$.

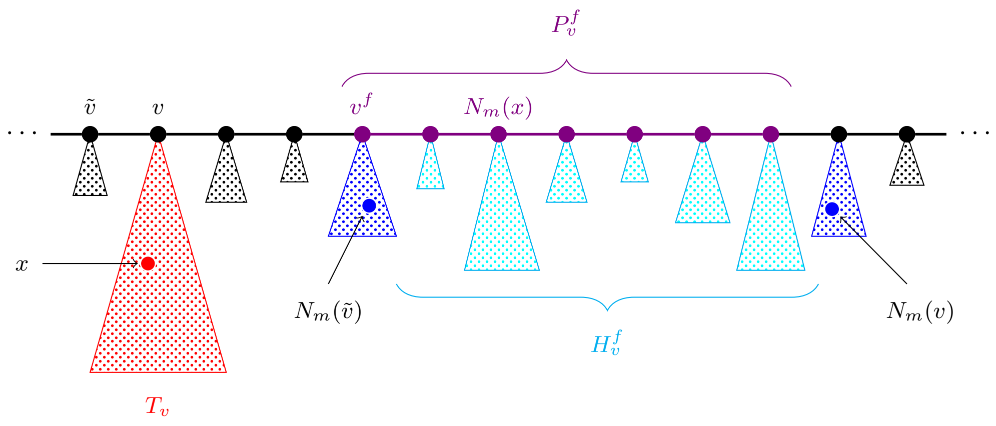
In this case, $N_m(v) \notin V_P$ and $N_m^{-1}(v) \notin V_P$ for all $v \in V_P$. Then, $\lvert V_P\rvert = \lvert N_m(V_P)\rvert \le \lvert V \setminus V_P\rvert$, which implies that $\lvert V_P\rvert \le \frac{1}{2}n$ and

$$d \le \tfrac{1}{2}. \tag{5.5}$$

A vertex $v \in V_P$ is called *b-special*, if there is a vertex $w \in T'_v$ with $N_m^{-1}(w) \in V_P$. We use b as in *backward*, because there is some vertex w in T_v such that going m steps backward from w in the numeration gives a vertex on P. A vertex $v \in V_P$ is called *f-special* if there is a vertex $w \in T'_v$ with $N_m(w) \in V_P$. We



a) A b-special vertex v and the sets P_v^b and H_v^b.



b) An f-special vertex v and the sets P_v^f and H_v^f.

Figure 5.5: Notation used in the proof of Lemma 5.7. A tree T_u with $u \in V_P$ is colored blue if the vertex u is b-special. A tree T_u with $u \in V_P$ is colored red if the vertex u is f-special.

use f as in *forward*, because there is some vertex w in T_v such that going m steps forward from w in the numeration gives a vertex on P.

For every vertex $v \in V_P$, define

$$P_v^b := N_m^{-1}(T'_v) \cap V_P \qquad \text{and}$$
$$P_v^f := N_m(T'_v) \cap V_P.$$

See Figure 5.5 for an example. Note that, for every $v \in V_P$, when $P_v^b \neq \emptyset$, then the vertices in P_v^b appear consecutively on the unique cycle in G^+ and P_v^b induces a path or a cycle in G^+. Moreover, P_v^b is not empty if and only if v is b-special. Furthermore, P_v^b and P_w^b are disjoint for distinct vertices $v, w \in V_P$ and every vertex in P is contained in a set P_v^b for some b-special $v \in V_P$. The same holds analogously for the sets P_v^f and the following proposition is obtained.

Proposition 5.9.

a) $\{P_v^b\colon\ v \in V_P$ *is b-special*$\}$ *is a partition of* V_P.

b) $\{P_v^f\colon\ v \in V_P$ *is f-special*$\}$ *is a partition of* V_P.

For each b-special $v \in V_P$, define $v^b = N_m^{-1}(x)$, where x is the smallest vertex in T'_v with $N_m^{-1}(x) \in V_P$.

Note that, for a b-special vertex $v \in V_P$, if the set P_v^b does not consist of all vertices in V_P, then v^b is one of the ends of the path that P_v^b induces in G^+. Similarly, for each f-special $v \in V_P$, define $v^f = N_m(x)$ where x is the smallest vertex in T_v' with $N_m(x) \in V_P$. Then, for each f-special $v \in V_P$ for which P_v^f does not induce a cycle in G^+, the vertex v^f is one of the leaves of the path that P_v^f induces in G^+. Furthermore, let

$$H_v^b := \bigcup_{x \in P_v^b \setminus \{v^b\}} T_x' \qquad \text{for all b-special } v \in V_P,$$

$$H_v^f := \bigcup_{x \in P_v^f \setminus \{v^f\}} T_x' \qquad \text{for all f-special } v \in V_P,$$

and $H_v^b = \emptyset$ for every $v \in V_P$ that is not b-special as well as $H_v^f = \emptyset$ for every $v \in V_P$ that is not f-special.

Lemma 5.10.

a) For every b-special $v \in V_P$, the vertex v^b is f-special.

b) For every f-special $v \in V_P$, the vertex v^f is b-special.

c) A vertex $w \in V_P$ is b-special if and only if there exists an f-special vertex $v \in V_P$ such that $w = v^f$.

d) A vertex $w \in V_P$ is f-special if and only if there exists a b-special vertex $v \in V_P$ such that $w = v^b$.

Proof.

a) Consider a b-special vertex $v \in V_P$ and let $\tilde{v} \in V_P$ be the vertex before v on the path P. As the vertex $N_m^{-1}(\tilde{v})$ is not in V_P by the assumption of Case 2, $N_m^{-1}(\tilde{v})$ must be in T_{v^b}', i. e., T_{v^b} contains the vertex $x := N_m^{-1}(\tilde{v})$, $x \neq v^b$, and $N_m(x) = \tilde{v} \in V_P$. Hence, v^b is f-special. See also Figure 5.5a).

b) Consider an f-special vertex $v \in V_P$ and let $\tilde{v} \in V_P$ be the vertex before v on the path P. As the vertex $N_m(\tilde{v})$ is not in V_P by the assumption of Case 2, $N_m(\tilde{v})$ must be in T_{v^f}', i. e., T_{v^f} contains the vertex $x = N_m(\tilde{v})$, $x \neq v^f$, and $N_m^{-1}(x) = \tilde{v} \in V_P$. Hence, v^f is b-special. See also Figure 5.5b).

c) First, if there is an f-special vertex $v \in V_P$ such that $w = v^f$, then Part b) shows that w is b-special. Second, as two distinct b-special vertices v and w have distinct vertices v^b and w^b and similarly for f-special vertices, it follows that

$$\begin{aligned} & \left|\left\{w \in V_P\colon\ w \text{ is b-special}\right\}\right| \le \left|\left\{w^b \in V_P\colon\ w \text{ is b-special}\right\}\right| \\ \overset{\text{Part a)}}{\le} & \left|\left\{\ v \in V_P\colon\ v \text{ is f-special}\ \right\}\right| \le \left|\left\{\ v^f \in V_P\colon\ v \text{ is f-special}\ \right\}\right| \\ \overset{\text{Part b)}}{\le} & \left|\left\{w \in V_P\colon\ w \text{ is b-special}\right\}\right|. \end{aligned}$$

Note that all inequalities in the above equation must be equalities, which shows that a vertex $w \in V_P$ can only be b-special if there exists an f-special vertex $v \in V_P$ such that $w = v^f$.

d) Analog to Part c). □

Lemma 5.10c) immediately implies that

$$\sum_{\substack{v \in V_P\colon \\ v \text{ is f-special}}} |T_{v^f}'| = \sum_{\substack{w \in V_P\colon \\ w \text{ is b-special}}} |T_w'| \tag{5.6}$$

and Lemma 5.10d) implies that

$$\sum_{\substack{v \in V_P\colon \\ v \text{ is b-special}}} |T_{v^b}'| = \sum_{\substack{w \in V_P\colon \\ w \text{ is f-special}}} |T_w'|. \tag{5.7}$$

For every b-special $v \in V_P$ and for every vertex $u \in P_v^b \cup H_v^b$, the vertex $N_m(u)$ lies in T'_v. Thus,

$$|T'_v| \;\geq\; |P_v^b| + |H_v^b| \qquad\qquad \text{for every b-special } v \in V_P. \tag{5.8}$$

Similarly, for every f-special $v \in V_P$ and for every vertex $u \in P_v^f \cup H_v^f$, the vertex $N_m^{-1}(u)$ lies in T'_v. Therefore,

$$|T'_v| \;\geq\; |P_v^f| + |H_v^f| \qquad\qquad \text{for every f-special } v \in V_P. \tag{5.9}$$

Note that, for a b-special vertex $v \in V_P$, the sets P_v^b, H_v^b, and T'_{v^b} are pairwise disjoint. Furthermore, Proposition 5.9a) implies that every vertex $x \in V$ is in exactly one set $P_v^b \,\dot\cup\, H_v^b \,\dot\cup\, T'_{v^b}$ for some b-special vertex $v \in V_P$. Consequently,

$$(1-d)n \;=\; |V \setminus V_P| \;=\; \sum_{\substack{v \in V_P :\\ v \text{ is b-special}}} \left(|T'_{v^b}| + |H_v^b|\right) \qquad \text{and} \tag{5.10}$$

$$dn \;=\; |V_P| \;=\; \sum_{\substack{v \in V_P :\\ v \text{ is b-special}}} |P_v^b|. \tag{5.11}$$

Similarly, Proposition 5.9b) implies that every vertex $x \in V$ is in exactly one set $P_v^f \,\dot\cup\, H_v^f \,\dot\cup\, T'_{v^f}$ for some f-special vertex $v \in V_P$ and therefore

$$(1-d)n \;=\; |V \setminus V_P| \;=\; \sum_{\substack{v \in V_P :\\ v \text{ is f-special}}} \left(|T'_{v^f}| + |H_v^f|\right) \qquad \text{and} \tag{5.12}$$

$$dn \;=\; |V_P| \;=\; \sum_{\substack{v \in V_P :\\ v \text{ is f-special}}} |P_v^f|. \tag{5.13}$$

Now, (5.10)-(5.13) together imply

$$\sum_{\substack{v \in V_P :\\ v \text{ is b-special}}} \left(|T'_{v^b}| + |H_v^b|\right) \;+\; \sum_{\substack{v \in V_P :\\ v \text{ is f-special}}} \left(|T'_{v^f}| + |H_v^f|\right) \;=\; 2\,|V \setminus V_P|$$

$$= 2\,\frac{1-d}{d}\,|V_P| \;=\; \frac{1-d}{d}\left(\sum_{\substack{v \in V_P :\\ v \text{ is b-special}}} |P_v^b| \;+\; \sum_{\substack{v \in V_P :\\ v \text{ is f-special}}} |P_v^f|\right).$$

Now, (5.6) and (5.7) allow to rearrange the terms in the sums in the first expression, such that both sums use $|T'_v|$ instead of $|T'_{v^b}|$ and $|T'_{v^f}|$, respectively. Thus,

$$\sum_{\substack{v \in V_P :\\ v \text{ is b-special}}} \left(|T'_v| + |H_v^b|\right) \;+\; \sum_{\substack{v \in V_P :\\ v \text{ is f-special}}} \left(|T'_v| + |H_v^f|\right)$$

$$= \frac{1-d}{d}\left(\sum_{\substack{v \in V_P :\\ v \text{ is b-special}}} |P_v^b| \;+\; \sum_{\substack{v \in V_P :\\ v \text{ is f-special}}} |P_v^f|\right).$$

As there is at least one b-special vertex $v \in V_P$ and at least one f-special vertex $v \in V_P$, the previous equation implies the following proposition.

Proposition 5.11.
At least one of the following exists:

a) a b-special vertex $v \in V_P$ such that $|T'_v| + |H_v^b| \leq \left(\frac{1}{d} - 1\right)|P_v^b|$, or

b) an f-special vertex $v \in V_P$ such that $|T'_v| + |H_v^f| \leq \left(\frac{1}{d} - 1\right)|P_v^f|$.

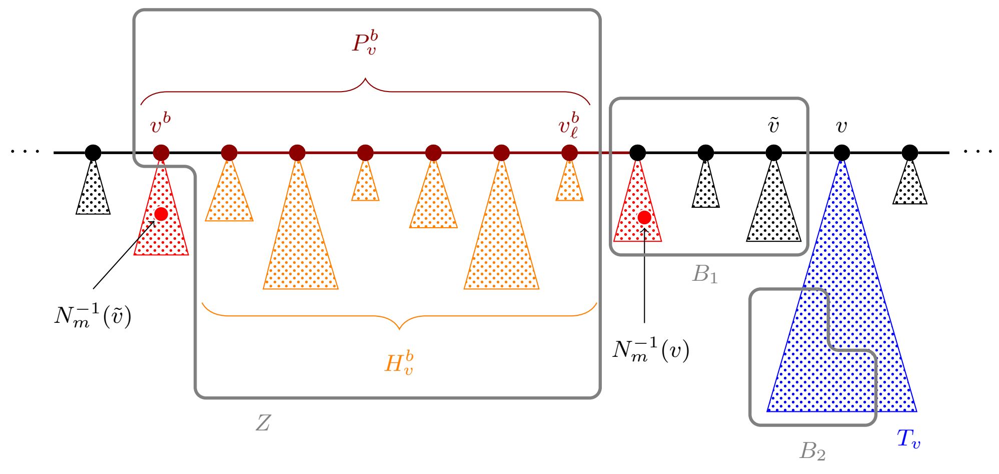


Figure 5.6: Proof of Lemma 5.7. Cut (B, W, Z) in Case 2a).

Case 2a) There is a b-special $v \in V_P$ with $|T'_v| + |H_v^b| \leq \left(\frac{1}{d} - 1\right)|P_v^b|$.

Replacing $|T'_v|$ with (5.8) yields

$$\begin{aligned} |P_v^b| + 2|H_v^b| \;&\leq\; \left(\tfrac{1}{d} - 1\right)|P_v^b| \qquad \Rightarrow \quad 2|P_v^b| + 2|H_v^b| \;\leq\; \tfrac{1}{d}\,|P_v^b| \\ \Rightarrow \quad |P_v^b| + |H_v^b| \;&\leq\; \tfrac{1}{2d}\,|P_v^b|. \end{aligned} \tag{5.14}$$

Next, a cut (B, W, Z) with the properties required for Option 2) in Lemma 5.7 is constructed. Define $Z := H_v^b \dot\cup P_v^b$, which satisfies $Z \neq \emptyset$ since v is b-special and P_v^b hence contains at least one vertex. Furthermore, Z and T'_v are disjoint. Indeed, assume that there was a vertex x in $Z \cap T'_v$. As $P_v^b \subseteq V_P$ and T'_v does not contain any vertex from V_P, it follows that $x \in H_v^b$. By the definition of H_v^b, the vertex v must be in P_v^b and $v \neq v^b$. Now, $v \in P_v^b = N_m^{-1}(T'_v) \cap V_P$ implies that T'_v contains a vertex y such that $N_m^{-1}(y) = v$. Now, any vertex $z \in T'_v$ that is smaller than y satisfies $N_m^{-1}(z) \in T'_v$. Therefore, v must be v^b, which is a contradiction and implies that Z and T'_v are disjoint. Together with (5.8) it follows that $|Z| \leq \frac{1}{2}n$. Furthermore, (5.14) implies that $|Z| \leq |H_v^b| + |P_v^b| \leq \frac{1}{2d}|P_v^b|$. As P_v^b induces a collection of paths in $G[Z]$ with at most one path in each component of $G[Z]$, the relative diameter of $G[Z]$ can be estimated by

$$\operatorname{diam}^*(G[Z]) \;\geq\; \frac{1}{|Z|}|P_v^b| \;\geq\; 2d.$$

Define $v_\ell^b = N_m^{-1}(x)$, where x is the largest vertex among all vertices in T'_v with $N_m^{-1}(x) \in V_P$. Note that, if $P_v^b \neq \{v^b\}$ and $P_v^b \neq V_P$, then v_ℓ^b is the leaf of the path induced by P_v^b in G^+ that is not v^b. Let $\tilde{v}$ be the vertex before v on P. If $v_\ell^b = \tilde{v}$, then define $B_1 := \emptyset$. Otherwise, define

$$B_1 := \left\{x \in V\colon\; x \text{ is between } v_\ell^b + 1 \text{ and } \tilde{v}\right\},$$

see Figure 5.6. Moreover, define $\tilde{m} := m - |B_1|$. Since $N_m(v_\ell^b)$ is in T'_v, it follows that $1 \leq \tilde{m} \leq |T'_v|$. Define $c := 2 - \frac{1}{1-d} = \frac{1-2d}{1-d}$ and note that $c \in [0, 1)$ by (5.5). Corollary 4.4 implies that the forest $G[T'_v]$ allows a c-approximate $\tilde{m}$-cut (B_2, W_2) of width[1]

$$e_{G[T'_v]}(B_2, W_2) \;\leq\; \left\lceil \frac{2c}{1-c} \right\rceil \Delta(G) \;=\; \left\lceil \frac{2(1-2d)}{d} \right\rceil \Delta(G)$$

[1]Note that Lemma 4.5b) provides a better bound on the width of a c-approximate $\tilde{m}$-cut in $G[T'_v]$. For now, we will use Corollary 4.4 as the computations are easier. The improvements will be discussed in Section 5.2.3.

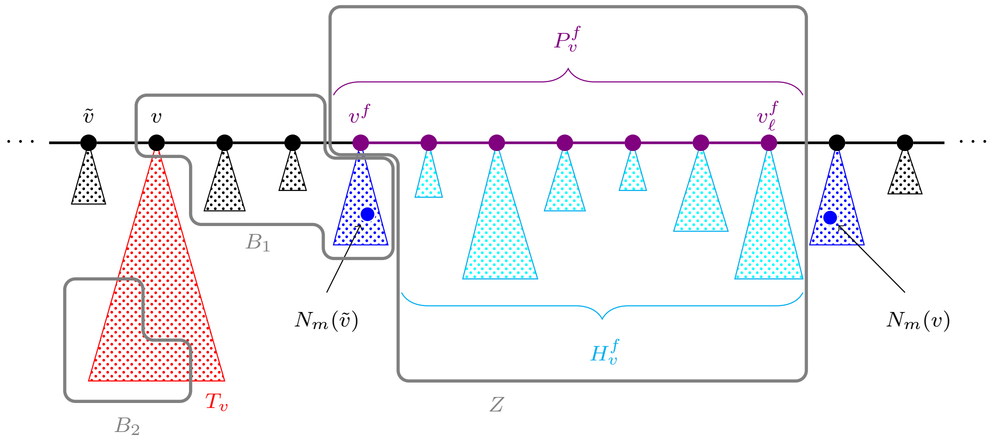


Figure 5.7: Proof of Lemma 5.7. Cut (B, W, Z) in Case 2b).

$$= \left\lceil \frac{2}{d} - 4 \right\rceil \Delta(G) \;\leq\; \left(\frac{2}{d} - 3\right) \Delta(G). \tag{5.15}$$

Note that $B_2 \,\dot\cup\, W_2 = T'_v$ and $c\tilde{m} \leq |B_2| \leq \tilde{m}$.

Define $B := B_1 \cup B_2$ and note that B_1 and B_2 are disjoint by construction. Moreover,

$$\begin{aligned} m - |B| \;&=\; (m - |B_1|) - |B_2| \;\leq\; \tilde{m} - c\tilde{m} \;\leq\; (1-c)|T'_v| \\ &\leq\; \frac{d}{1-d}|T'_v| \;\leq\; |P_v^b| \;\leq\; |Z|, \end{aligned}$$

where the second to last inequality holds by the assumption $|T'_v| + |H_v^b| \leq \left(\frac{1}{d} - 1\right)|P_v^b|$ of Case 2a). Hence, $|B| \leq m \leq |B| + |Z|$. Since $B_2 \subseteq T'_v$ and the sets T'_v and Z are disjoint as argued above, the sets B and Z are disjoint.

Let $W := V \setminus (Z \cup B)$. Next, the width of the cut (B, W, Z) is estimated. At most $\Delta(G) + 1$ edges of G are cut by $(Z, B \cup W)$, i. e., at most all edges incident to v^b and the edge after v_ℓ^b on P. Moreover, at most $e_{G[T'_v]}(B_2, W_2) + \Delta(G)$ edges are cut by (B, W) in $G[B \cup W]$, which are the edges cut by (B_2, W_2) within $G[T'_v]$ and all edges incident to v (if $v \notin Z$, i. e., if $v \neq v^b$). With (5.15) it follows that

$$\begin{aligned} e_G(B, W, Z) \;&\leq\; e_G(Z, B \cup W) + e_{G[B \cup W]}(B, W) \;\leq\; \Delta(G) + 1 + e_{G[T'_v]}(B_2, W_2) + \Delta(G) \\ &\leq\; 3\Delta(G) + \left(\frac{2}{d} - 3\right)\Delta(G) \;\leq\; \frac{2}{d}\,\Delta(G). \end{aligned}$$

Case 2b) There is an f-special $v \in V_P$ with $|T'_v| + |H_v^f| \leq \left(\frac{1}{d} - 1\right)|P_v^f|$.

This case is similar to Case 2a) but not completely analogous as we cannot simply reverse the labeling to obtain the situation of Case 2a), because this does not yield a P-labeling. Analogously to Case 2a), $|P_v^f| + |H_v^f| \leq \frac{1}{2d}|P_v^f|$ can be derived by replacing $|T'_v|$ with (5.9). Furthermore, define $Z := H_v^f \,\dot\cup\, P_v^f$ and deduce analogously to Case 2a) that $\operatorname{diam}^*(G[Z]) \geq 2d$ and $0 < |Z| \leq \frac{1}{2}n$, as Z and T'_v are disjoint. Note that v is between $N_m^{-1}(v^f)$ and v^f. Moreover, $v = v^f$ is possible and, in this case, v lies in Z, and otherwise v does not lie in Z. Therefore, the set B_1 is defined in a slightly different way than in Case 2a). Let

$$B_1 := \left\{x \in V \colon\; x \text{ is between } v \text{ and } v^f,\, x \neq v^f\right\}$$

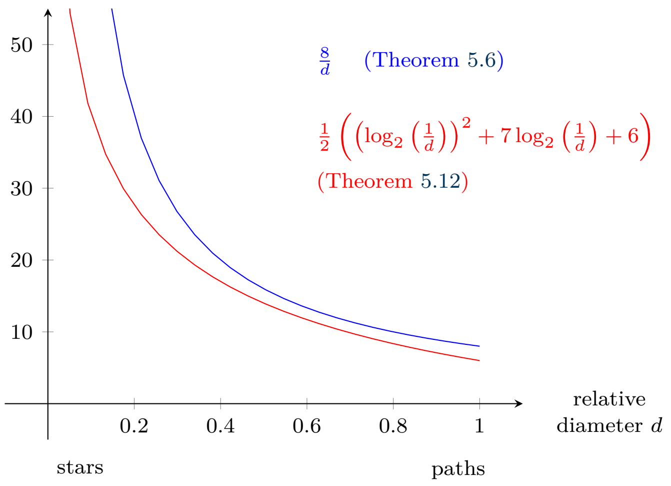


Figure 5.8: Comparing the bounds provided by Theorem 5.6 and Theorem 5.12. The factor $\Delta(G)$, which is present in both bounds, is ignored.

and $\tilde{m} := m - |B_1|$. See also Figure 5.7 for a visualization. Since $N_m^{-1}(v^f) \in T'_v$ and $v \in B_1$ or $v = v^f \in Z$, it follows that $|T'_v| \geq \tilde{m}$. Let $c := 2 - \frac{1}{1-d}$ as in Case 2a). Analogously to Case 2a), Corollary 4.4 implies that the forest $G[T'_v]$ admits a c-approximate $\tilde{m}$-cut (B_2, W_2) of width $e_{G[T'_v]}(B_2, W_2) \leq \left(\frac{2}{d} - 3\right)\Delta(G)$. Defining $B := B_1 \dot{\cup} B_2$, one can argue that $|B| \leq m \leq |B| + |Z|$ and that B is disjoint from Z. Let $W := V \setminus (B \cup Z)$. Then, at most $\Delta(G) + 1$ edges are cut by $(Z, B \dot{\cup} W)$, i. e., the edges incident to v^f and the edge after the other end v_ℓ^f of P_v^f. At most $e_{G[T'_v]}(B_2, W_2) + \Delta(G)$ edges are cut by (B, W) in $G[B \cup W]$, which are the edges cut by (B_2, W_2) within $G[T'_v]$ and all edges incident to v. All together, $e_G(B, W, Z) \leq \frac{2}{d}\Delta(G)$ is obtained.

5.2.3 Improving the Bound on the Width of the Cut

After presenting a proof of Theorem 5.6 in the previous two subsections, this subsection is devoted to improving its bound on the width of the cut. The first improvement arises from using Lemma 4.5b) instead of Corollary 4.4 in the proof of Lemma 5.7. This results in an improvement of the width of the cut in Option 2) in Lemma 5.7. Furthermore, the induction from the proof of Theorem 5.6 is replaced by an iterative procedure that relies on the improved version of Lemma 5.7. This will result in the following version of Theorem 5.6.

Theorem 5.12 (improved version of Theorem 5.6).
For every forest G on n vertices and for every $m \in [n]$, there is an m-cut (B, W) in G with

$$e_G(B, W) \;\leq\; \frac{1}{2}\,\Delta(G)\left(\left(\log_2\left(\frac{1}{\operatorname{diam}^*(G)}\right)\right)^2 + 7\log_2\left(\frac{1}{\operatorname{diam}^*(G)}\right) + 6\right).$$

Figure 5.8 visualizes the bounds provided by Theorem 5.6 and Theorem 5.12. Observe also that Theorem 5.1 provides a much smaller bound for forests with relative diameter greater than $\frac{1}{4}$.

Let us now discuss the details of the improvements. First, replace Corollary 4.4 by Lemma 4.5b). Recall the proof of Lemma 5.7 and, in particular, the computation of the upper bound on $e_G(B, W, Z)$ in Case 2a) on Page 162. There, we defined $c := \frac{1-2d}{1-d}$ and used Corollary 4.4 to find a c-approximate

$\tilde{m}$-cut (B_2, W_2) in $G[T'_v]$ of width at most

$$e_{G[T'_v]}(B_2, W_2) \;\leq\; \left\lceil \frac{2c}{1-c} \right\rceil \Delta(G) \;\leq\; \left(\frac{2}{d} - 3\right) \Delta(G).$$

Lemma 4.5b) implies that there is a c-approximate $\tilde{m}$-cut (B_2, W_2) in $G[T'_v]$ of width at most

$$e_{G[T'_v]}(B_2, W_2) \;\leq\; \left\lceil \log_2\left(\frac{1}{1-c}\right) \right\rceil \Delta(G) \;\leq\; \left\lceil \log_2\left(\frac{1-d}{d}\right) \right\rceil \Delta(G) \;\leq\; \left(\log_2\left(\frac{1}{d}\right) + 1\right) \Delta(G).$$

Also, when estimating the width of the cut $(Z, B \mathbin{\dot\cup} W)$, one can be a little more careful. If $v^b = v^b_\ell$, then $e_G(Z, B \mathbin{\dot\cup} W) \leq \Delta(G)$ as only edges incident to v^b are cut. Otherwise, note that v^b is f-special by Lemma 5.10a) and, thus, $T'_{v^b} \neq \emptyset$. Recall that y_0 is the leaf of the path P that received label n and that y_0 is a leaf of G. Therefore, $T'_{y_0} = \emptyset$ and $v^b \neq y_0$, which implies that the edge after v^b on P exists. Then, the cut $(Z, B \mathbin{\dot\cup} W)$ cuts at most $\Delta(G)$ edges, which are the edge after v^b_ℓ on the path P and all edges incident to v^b except the edge after v^b on the path P. Consequently, $e_G(Z, B \mathbin{\dot\cup} W) \leq \Delta(G)$. Recalling that $e_{G[B \cup W]}(B, W) \leq \Delta(G) + e_{G[T'_v]}(B_2, W_2)$, the following improved bound is obtained

$$\begin{aligned} e_G(B, W, Z) \;&\leq\; e_G(Z, B \mathbin{\dot\cup} W) + e_{G[B \cup W]}(B, W) \\ &\leq\; 2\Delta(G) + \left(\log_2\left(\tfrac{1}{d}\right) + 1\right)\Delta(G) \;\leq\; \Delta(G) \cdot \log_2\left(\tfrac{8}{d}\right). \end{aligned}$$

Similarly, in Case 2b) in the proof of Lemma 5.7, using Lemma 4.5b) instead of Corollary 4.4 implies that $e_{G[T'_v]}(B_2, W_2) \leq \left(\log_2\left(\frac{1}{d}\right) + 1\right)\Delta(G)$. Furthermore, the width of the cut $(Z, B \mathbin{\dot\cup} W)$ in G can be estimated by $e_G(Z, B \mathbin{\dot\cup} W) \leq \Delta(G)$ as done above for Case 2a). Then, the improved bound

$$e_G(B, W, Z) \;\leq\; \Delta(G) \cdot \log_2\left(\tfrac{8}{d}\right) \tag{5.16}$$

for the cut (B, W, Z) in G is obtained. Consequently, the width of the cut in Option 2) in Lemma 5.7 can be improved such that the following lemma is obtained.

Lemma 5.13 (improved version of Lemma 5.7).
For all forests G on n vertices and for all $m \in [n]$, there is a cut (B, W, Z) in G such that one of the following two options is satisfied:

1) $Z = \emptyset$, $|B| = m$, and $e_G(B, W, Z) \leq 2$, or

2) $Z \neq \emptyset$, $|Z| \leq \frac{1}{2}n$, $|B| \leq m \leq |B| + |Z|$, $e_G(B, W, Z) \leq \log_2\left(\frac{8}{\mathrm{diam}^*(G)}\right)\Delta(G)$, and $\mathrm{diam}^*(G[Z]) \geq 2\,\mathrm{diam}^*(G)$.

This lemma is now used to prove Theorem 5.12. Instead of applying induction as in the proof of Theorem 5.6, an iterative procedure, that follows the same idea, is used.

Proof of Theorem 5.12. Let $G = (V, E)$ be an arbitrary forest on n vertices, fix an $m \in [n]$, and define $d := \mathrm{diam}^*(G)$. If $\Delta(G) \leq 2$, then Lemma 5.2b) guarantees that G allows an m-cut of width at most $\Delta(G)$ and there is nothing else to show. So, from now on, assume that $\Delta(G) \geq 3$ and apply the procedure described in Algorithm 5.1 to compute an m-cut in G.

The procedure described in Algorithm 5.1 always terminates because G is reset to $G[\tilde{Z}]$ and, thus, the number of vertices of G decreases in each execution of the while loop due to Lemma 5.13. Denote by s^* the number of executions of the while loop. To state some invariants, denote by G_s and B_s the state of the graph G and the set B after the s^{th} execution of the while loop for every $s \in [s^*]$ and let G_0 be the input graph and $B_0 := \emptyset$, which are also the states of the variables B and G before the first execution of the while loop. For every $s \in [s^*] \cup \{0\}$, define $n_s := |V(G_s)|$. Furthermore, denote by $\tilde{B}_s$, $\tilde{W}_s$, and $\tilde{Z}_s$ the sets $\tilde{B}$, $\tilde{W}$, and $\tilde{Z}$ used in the s^{th} execution of the while loop, respectively. Then, for every $s \in [s^*] \cup \{0\}$

Algorithm 5.1: Computes an m-cut in a forest G.

```
   Input: forest $G = (V, E)$ on $n$ vertices, $m \in [n]$.
   Output: $m$-cut $(B, W)$ in $G$.
1  $B \leftarrow \emptyset$, $V_0 \leftarrow V$;
2  While $|B| < m$ do
3      Apply Lemma 5.13 to $G$ with size-parameter $\tilde{m} = m - |B|$ to obtain a cut $(\tilde{B}, \tilde{W}, \tilde{Z})$ in $G$;
4      $B \leftarrow B \cup \tilde{B}$;
5      $G \leftarrow G[\tilde{Z}]$;
6  Endw
7  Return $(B, V_0 \setminus B)$;
```

(i) if $s \neq s^*$, then $0 < m - |B_s| \leq n_s$,
(ii) $B_s \cap V(G_s) = \emptyset$ and, in particular for $s \neq 0$, $B_{s-1} \cap \tilde{B}_s = \emptyset$,
(iii) if $s \neq s^*$, then $\operatorname{diam}^*(G_s) \geq 2^s d$,
(iv) $n_s \leq \frac{1}{2^s} n_0$, and
(v) if $s \neq 0$, then $e_{G_{s-1}}(\tilde{B}_s, \tilde{W}_s, \tilde{Z}_s) \leq \log_2\left(\frac{8}{\operatorname{diam}^*(G_{s-1})}\right) \Delta(G)$.

All invariants can be shown inductively with Lemma 5.13. Note that the application of Lemma 5.13 in Line 3 is feasible by (i) and that the union in Line 4 is a disjoint union by (ii). Recall that $\operatorname{diam}^*(H) \leq 1$ for all forests H. Therefore, when $\operatorname{diam}^*(G_s) > \frac{1}{2}$, then Option 2) in Lemma 5.13 is infeasible and Option 1) must occur, which means that the last execution of the while loop is reached. Consequently,

$$s^* \leq \log_2\left(\tfrac{1}{d}\right) + 1 \tag{5.17}$$

because otherwise

$$\operatorname{diam}^*(G_{s^*-1}) \overset{\text{(iii)}}{\geq} 2^{s^*-1}\, d \;>\; 2^{\log_2 \frac{1}{d}}\, d \;=\; 1.$$

Furthermore, the returned set B satisfies $B = \bigcup_{s=1}^{s^*} \tilde{B}_s$ and, therefore,

$$\begin{aligned}
e_G(B, W) &\leq \sum_{s=1}^{s^*} e_{G_{s-1}}(\tilde{B}_s, \tilde{W}_s, \tilde{Z}_s) \overset{\text{(v),(iii)}}{\leq} \Delta(G) \sum_{s=1}^{s^*} \log_2\left(\frac{8}{2^{s-1} d}\right) \\
&\leq \Delta(G) \sum_{s=1}^{s^*} \left(\log_2\left(\tfrac{8}{d}\right) - (s-1)\right) \leq \Delta(G)\left(s^* \log_2\left(\tfrac{8}{d}\right) - \sum_{s=1}^{s^*}(s-1)\right) \\
&\leq \Delta(G)\left(s^* \log_2\left(\tfrac{8}{d}\right) - \tfrac{1}{2} s^*(s^*-1)\right) \leq \Delta(G) s^* \left(\log_2\left(\tfrac{8}{d}\right) + \tfrac{1}{2} - \tfrac{1}{2} s^*\right).
\end{aligned}$$

The last term is a quadratic function in s^*, whose maximum value is achieved at $s^* = \log_2\left(\frac{8}{d}\right) + \frac{1}{2}$, which is larger than the upper bound on s^* in (5.17). Consequently,

$$\begin{aligned}
e_G(B, W) &\leq \Delta(G)\left(\log_2\left(\tfrac{1}{d}\right) + 1\right)\left(\log_2\left(\tfrac{1}{d}\right) + \tfrac{7}{2} - \tfrac{1}{2}\log_2\left(\tfrac{1}{d}\right) - \tfrac{1}{2}\right) \\
&\leq \tfrac{1}{2}\Delta(G)\left(\log_2\left(\tfrac{1}{d}\right) + 1\right)\left(\log_2\left(\tfrac{1}{d}\right) + 6\right) \\
&\leq \tfrac{1}{2}\Delta(G)\left(\left(\log_2\left(\tfrac{1}{d}\right)\right)^2 + 7\log_2\left(\tfrac{1}{d}\right) + 6\right),
\end{aligned}$$

which completes the proof. □

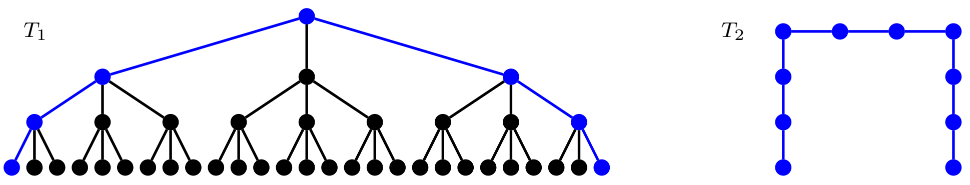


Figure 5.9: Example showing that it is not possible to work with only one component when implementing the algorithm contained in Theorem 5.14. In each component, a longest path is colored blue.

5.2.4 Linear-Time Algorithm for Trees

This section discusses the algorithm contained in Theorem 1.8, which is a linear-time algorithm that computes an exact cut with the properties in Theorem 5.12 in a forest.

Theorem 5.14 (algorithmic version of Theorem 5.12, Theorem 1.8 restated).
For every forest G on n vertices, an m-cut (B, W) of width at most

$$e_G(B,W) \;\le\; \frac{1}{2}\,\Delta(G)\left(\left(\log_2\left(\frac{1}{\operatorname{diam}^*(G)}\right)\right)^2 + 7\log_2\left(\frac{1}{\operatorname{diam}^*(G)}\right) + 6\right)$$

can be computed in $\mathcal{O}(n)$ time.

Remark 5.15.
The previous theorem also implies that a cut with the properties in Theorem 5.6 can be computed in $\mathcal{O}(n)$ time as the constructions of the cut in Theorem 5.6 and Theorem 5.12 are identical. Furthermore, it is not hard to adapt the algorithm to compute a cut with the properties in Theorem 5.1 or Lemma 5.3.

Consider the algorithm contained in the previous theorem. When the input forest is not connected, it is not sufficient to work with one component as we did in the proof of Lemma 4.1 for example. Indeed, consider a forest G that is composed of $\ell \ge 2$ components $T_1, T_2, \ldots, T_\ell$. The construction from Section 5.2.1-5.2.3 works with the relative diameter of G. Denote by n the number of vertices of G and let n_h be the number of vertices of T_h for all $h \in [\ell]$. Note that

$$\operatorname{diam}^*(G) \;=\; \frac{1}{n}\sum_{h\in[\ell]} n_h \operatorname{diam}^*(T_h)$$

and, hence, there is an $h \in [k]$ with $\operatorname{diam}^*(T_h) \ge \operatorname{diam}^*(G)$. However, it might not be possible to distribute the vertex sets of the other components to the sets B and W of an exact cut (B, W) in G such that (B, W) cuts only edges within T_h. For a concrete example, consider the forest G on 50 vertices that is composed of a perfect ternary tree T_1 on 40 vertices, i. e., a perfect ternary tree of height three, and a path T_2 on 10 vertices, see Figure 5.9. Then, $\operatorname{diam}^*(G) = \frac{1}{50}(7 + 10) = 0.34$, $\operatorname{diam}^*(T_1) = \frac{7}{40} = 0.175$, and $\operatorname{diam}^*(T_2) = \frac{10}{10} = 1$. However, there is no bisection in G, which cuts only edges within T_2, as T_1 contains more than half of the vertices of G.

When proving Theorem 5.14, we will follow the construction presented in Section 5.2.2 and Section 5.2.3, where only trees were considered due to Lemma 5.2. The reason for doing so is that the notation simplifies when the considered forest is connected. However, the method would also work for a disconnected forest when considering a collection of paths that contains one longest path for each component of the forest instead of one longest path in the given tree. In order to use the simpler notation and to closely follow the construction, the following algorithmic version of Lemma 5.2 is derived.

Lemma 5.16 (algorithmic version of Lemma 5.2).
For every forest G on n vertices and for every $m \in [n]$ the following can be computed in $\mathcal{O}(n)$ time
a) if $\Delta(G) \geq 3$, a tree T on n vertices with $G \subseteq T$, $\Delta(T) = \Delta(G)$, and $\operatorname{diam}^*(T) = \operatorname{diam}^*(G)$ or
b) if $\Delta(G) \leq 2$, an m-cut of width at most $\Delta(G)$ in G.

To prove Part a) of the previous lemma, one needs to compute a longest path in each component of the considered forest. Dijkstra described the procedure presented in Algorithm 5.2 to compute a longest path in a tree in linear time, see also [Bul+02]. There, for two vertices v, w of the input tree, $\operatorname{dist}(v, w)$ denotes the length of the unique v,w-path in the input tree. A proof of correctness for Algorithm 5.2 is presented in this section as this procedure is generalized later in Section 5.3.3 to compute a heaviest path in a tree decomposition. Observe that the brute-force approach to compute the distance between every pair of vertices takes quadratic time, even in the simple case of an unweighted tree.

Algorithm 5.2: Computes a longest path in a tree.

```
Input: tree $T$ on $n$ vertices.
Output: a longest path $P \subseteq T$ as an ordered list.
1 If $n = 1$ then
2     Return $P = T$.
3 Endif
4 Root $T$ at an arbitrary vertex $r$;
5 Find a leaf $s$ of $T$ with $\operatorname{dist}(r,s) \geq \operatorname{dist}(r,s')$ for all leaves $s'$ of $T$;
6 Root $T$ at $s$;
7 Find a leaf $t$ of $T$ with $\operatorname{dist}(s,t) \geq \operatorname{dist}(s,t')$ for all leaves $t'$ of $T$;
8 Return the unique $s$,$t$-path in $T$.
```

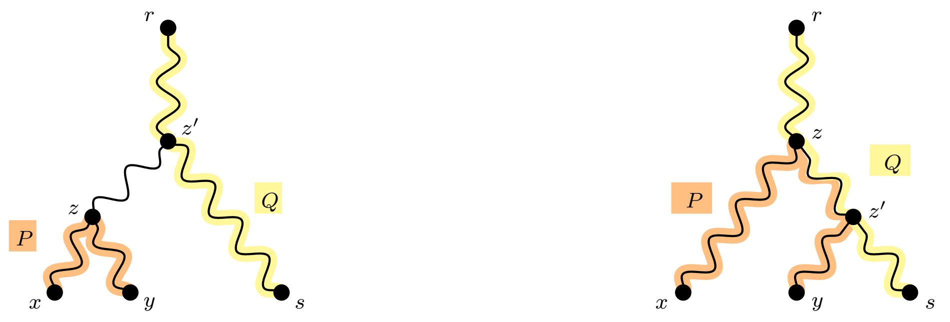
Lemma 5.17.
Algorithm 5.2 computes a longest path in the input tree T in $\mathcal{O}(n)$ time, where n denotes the number of vertices of T.

Proof. Let $T = (V, E)$ be an arbitrary tree on n vertices. First, it is shown that, when applied to T, Algorithm 5.2 computes a longest path in T. If $n = 1$, then T itself is a path and clearly Algorithm 5.2 returns a longest path. So, from now on, assume that $n \geq 2$. Let P be a longest path in T and denote by x and y the two leaves of P. Note that x and y must be leaves of T as well, as otherwise the path P could be extended. Let r, s, and t be the vertices used in Algorithm 5.2 when applied to the tree T and let Q be the unique r,s-path in T. For any three vertices $u, v, w \in V$, if v is on the unique u,w-path in T, then $\operatorname{dist}(u, w) = \operatorname{dist}(u, v) + \operatorname{dist}(v, w)$, and otherwise $\operatorname{dist}(u, w) < \operatorname{dist}(u, v) + \operatorname{dist}(v, w)$.

For a contradiction, assume that P and Q have no common vertex. Let z be the unique vertex in P with $\operatorname{dist}(r, z) \leq \operatorname{dist}(r, v)$ for all $v \in V(P)$. Furthermore, let z' be the first vertex on the path from z to r that is on Q, and note that $z' \notin V(P)$. See Figure 5.10a) for a visualization. The choice of s in Line 5 implies that $\operatorname{dist}(r, s) \geq \operatorname{dist}(r, y)$. As z' is on Q as well as on the unique r,y-path in T, it follows that $\operatorname{dist}(z', s) \geq \operatorname{dist}(z', y)$. The unique path from x to s in T uses the vertex z'. Therefore,

$$\operatorname{dist}(x, s) \;=\; \operatorname{dist}(x, z') + \operatorname{dist}(z', s) \;\geq\; \operatorname{dist}(x, z') + \operatorname{dist}(z', y) \;>\; \operatorname{dist}(x, y),$$

where the last inequality is strict because z' is not on P. Hence, $\operatorname{dist}(x, s) > \operatorname{dist}(x, y)$ and this contradicts that P is a longest path. Consequently, P and Q have at least one common vertex.



a) Notation if P and Q do not have a common vertex. **b)** Notation if P and Q have a common vertex.

Figure 5.10: Proof of Lemma 5.17.

Let z and z' be the first and the last vertex in Q, respectively, that are in P when traversing Q from r to s. Without loss of generality, we may assume that the path P when traversed from x to y uses first the vertex z and then the vertex z'. Therefore, z' is on the unique r,y-path in T. See also Figure 5.10b) for a visualization. The choice of s in Line 5 implies that $\operatorname{dist}(r, s) \geq \operatorname{dist}(r, y)$. As z' is on Q and on the unique r,y-path in T, it follows that $\operatorname{dist}(z', s) \geq \operatorname{dist}(z', y)$. Now, z' being on the unique x,s-path in T implies that

$$\operatorname{dist}(x, s) \;\geq\; \operatorname{dist}(x, z') + \operatorname{dist}(z', s) \;\geq\; \operatorname{dist}(x, z') + \operatorname{dist}(z', y) \;\geq\; \operatorname{dist}(x, y).$$

Consequently, the unique x,s-path in T is a longest path in T. So, there is a longest path in T that starts in s and the choice of t in Line 7 implies that the unique s,t-path in T is a longest path in T.

To complete the proof, the running time of Algorithm 5.2 is estimated. By Lemma 2.33, Line 4 and Line 6 each take $\mathcal{O}(n)$ time. To find a leaf that is furthest away from the root of a tree, the algorithm can use the vertex discovered last with a breadth-first traversal that was started at the root. Therefore, Line 5 and Line 7 each take $\mathcal{O}(n)$ time. A list of the vertices on the unique s,t-path in T can be computed by following the path from t up to the root r, which takes at most $\mathcal{O}(n)$ time. All in all, each step can be executed in $\mathcal{O}(n)$ time and the desired running time is achieved. □

Proof of Lemma 5.16. Recall the proof of Lemma 5.2. The algorithm presented here follows the same construction. Let G be an arbitrary forest on n vertices and let ℓ be the number of components of G. Denote by $T_1, \dots, T_\ell$ the components of G. Furthermore, for every $h \in [\ell]$, let n_h be the number of vertices of T_h. First, the algorithm determines the maximum degree of G by traversing its adjacency lists, which takes $\mathcal{O}(\|G\|) = \mathcal{O}(n)$ time. Afterwards, it determines the number ℓ as well as the numbers n_h for every $h \in [\ell]$, which takes $\mathcal{O}(\|G\|) = \mathcal{O}(n)$ time according to Lemma 2.25.

a) Assume that $\Delta(G) \geq 3$. If $\ell = 1$, the algorithm returns $T = G$. Otherwise, for every $h \in [\ell]$, the algorithm computes a longest path $P_h \subseteq T_h$ and its ends x_h and y_h. For each $h \in [\ell]$, this takes $\mathcal{O}(n_h)$ time by Lemma 5.17 and, hence, this takes $\mathcal{O}(n)$ time for all components together. Since, for every $h \in [\ell - 1]$, the edge $\{y_h, x_{h+1}\}$ can be added to G in constant time, the overall running time is $\mathcal{O}(n)$.

b) Assume that $\Delta(G) \leq 2$ and recall that T_h is a path for every $h \in [\ell]$. The set B will be returned as an unordered list L_B of vertices. Traversing the paths successively and adding the vertices to the list L_B until it contains exactly m vertices gives the desired cut in $\mathcal{O}(n)$ time. □

Recall that the construction used to prove Lemma 5.7 in Section 5.2.2 uses a P-labeling, where P is a longest path in the input tree. Using the general assumption that the input tree T satisfies $V(T) = [n]$ for some integer n, a P-labeling can be stored in two integer arrays A_L and A_V of length n. More precisely, for $v \in V(G)$, the entry $A_L[v]$ is the label of the vertex v and, for $\ell \in [n]$, the entry $A_V[\ell]$ is the vertex of G that received label ℓ. The next lemma says that such arrays can be computed in linear time.

Lemma 5.18.
For every tree T on n vertices and every path $P \subseteq T$, a P-labeling of the vertices of T and the path-vertex for every vertex $x \in V(T)$ can be computed in $\mathcal{O}(n)$ time.

Proof. Let T be an arbitrary tree on n vertices and let P be a path in T. Denote by x_0 and y_0 the ends of P. If P is given as a graph and not as a list, the algorithm first determines a list $(v_0 = x_0, v_1, \ldots, v_\ell = y_0)$ of the vertices on P in the order in which they appear on P, which takes $\mathcal{O}(n)$ time. Then, for each $h \in [\ell]$, the algorithm traverses the adjacency list of v_h and moves v_{h-1} to the beginning of the adjacency list of v_h, which takes at most $\mathcal{O}\left(\sum_{h\in[\ell]} \deg(v_h)\right) = \mathcal{O}(n)$ time. Using these reordered adjacency lists, the algorithm traverses T with a depth-first search starting at y_0. Observe that the vertices in P turn gray in the order $v_\ell = y_0, v_{\ell-1}, \ldots, v_0 = x_0$ and turn black in the order $v_0 = x_0, v_1, \ldots, v_\ell = y_0$. During the depth-first traversal, the algorithm labels each vertex when it turns black. While doing so, the path vertices are computed by keeping track of the vertex in P that is the last one that turned gray and is not yet black. To keep track of such a vertex, the algorithm uses a stack, where it pushes each vertex of P when it turns gray and pops the top vertex when it turns black. The extra time needed at each vertex is constant and, therefore, the entire procedure takes $\mathcal{O}(n)$ time. □

Lemma 5.19 (algorithmic version of Lemma 5.13).
For every forest G on n vertices and every $m \in [n]$, a cut (B, W, Z) in G satisfying one of the following options

1) $Z = \emptyset$, $|B| = m$, and $e_G(B, W, Z) \le 2$, or

2) $Z \ne \emptyset$, $|Z| \le \frac{1}{2}n$, $|B| \le m \le |B| + |Z|$, $e_G(B, W, Z) \le \log_2\left(\frac{8}{\operatorname{diam}^*(G)}\right)\Delta(G)$, and $\operatorname{diam}^*(G[Z]) \ge 2\operatorname{diam}^*(G)$

can be computed in $\mathcal{O}(n)$ time. The sets B, W, and Z are output as unordered lists of vertices.

Proof. Let G be an arbitrary forest on n vertices and fix an $m \in [n]$. We follow the construction presented in the proof of Lemma 5.7 and use the same notation. Recall that the improvement in the bound on the width of the cut in Option 2) presented in Section 5.2.3 is due to a tighter analysis and did not require to modify the construction. First, Lemma 5.16 implies that we may assume that G is a tree as otherwise there is a cut satisfying Option 1) or there is a tree with the same vertex set, the same maximum degree, and the same relative diameter as G, and both can be computed in $\mathcal{O}(n)$ time. The construction of the cut (B, W, Z) with the desired properties is summarized in Algorithm 5.3. It uses the following additional definitions

$$g^b(v) := \frac{|T'_v| + |H^b_v|}{|P^b_v|} \qquad \text{for every b-special } v \in V_P \text{ and}$$

$$g^f(v) := \frac{|T'_v| + |H^f_v|}{|P^f_v|} \qquad \text{for every f-special } v \in V_P,$$

as well as $g^b(v) := \infty$ for all $v \in V_P$ that are not b-special and $g^f(v) := \infty$ for all $v \in V_P$ that are not f-special. Furthermore, in Line 10, a vertex $v \in V_P$ is called a *doubling vertex* if v satisfies one of the following

a) v is b-special and $g^b(v) \le g^b(w)$ as well as $g^b(v) \le g^f(w)$ for all $w \in V_P$, or

Algorithm 5.3: Computes a cut (B, W, Z) with the properties in Lemma 5.19.

```
Input: tree $G = (V, E)$ on $n$ vertices, an integer $m \in [n]$.
Output: cut $(B, W, Z)$ with the properties stated in Lemma 5.19.
1  Compute a longest path $P = (V_P, E_P)$ in $G$ and let $L_P$ be a list of the vertices on $P$;
2  $d \leftarrow \mathrm{diam}^*(G)$;
3  Compute a $P$-labeling of the vertices in $G$ and the path-vertex for every vertex of $G$.
    Denote by $L(v)$ the label of the vertex $v \in V$ and by $L^{-1}(\ell)$ the vertex that received label $\ell \in [n]$;
4  If there is a $v \in V_P$ with $v \in V_P$ and $L^{-1}(L(v) + m) \in V_P$ then
5      Let $v$ be a vertex with $v \in V_P$ and $L^{-1}(L(v) + m) \in V_P$;
6      $B \leftarrow \{w\colon L(w)$ is between $L(v) + 1$ and $L(v) + m\}$,   $Z \leftarrow \emptyset$;
7  Else
8      For all $v \in V_P$, compute $p^b(v) := |P^b_v|$, $p^f(v) := |P^f_v|$, $h^b(v) := |H^b_v|$, and $h^f(v) := |H^f_v|$;
9      For all $v \in V_P$, compute $g^b(v)$ and $g^f(v)$;
10     Determine a doubling vertex $v \in V_P$;
11     If $g^b(v) \leq g^f(v)$ then
12         $Z \leftarrow H^b_v \cup P^b_v$;
13         Determine the vertex $v^b_\ell$ and let $\tilde{v}$ be the vertex before $v$ on $P$;
14         If $v^b_\ell = \tilde{v}$ then $B_1 \leftarrow \emptyset$ else $B_1 \leftarrow \{x \in V\colon L(x)$ is between $L(v^b_\ell) + 1$ and $L(\tilde{v})\}$;
15     Else
16         $Z \leftarrow H^f_v \cup P^f_v$;
17         Determine the vertex $v^f$;
18         $B_1 \leftarrow \{x \in V\colon L(x)$ is between $L(v)$ and $L(v^f)$, $x \neq v^f\}$;
19     Endif
20     $c \leftarrow 2 - \frac{1}{1-d}$,   $\tilde{m} \leftarrow m - |B_1|$;
21     Let $(B_2, W_2)$ be a $c$-approximate $\tilde{m}$-cut in $G[T'_v]$ with $e_{G[T'_v]}(B_2, W_2) \leq \left(\log_2\left(\frac{1}{d}\right) + 1\right)\Delta(G)$;
22     $B \leftarrow B_1 \cup B_2$;
23 Endif
24 $W \leftarrow V \setminus (B \cup Z)$;
25 Return $(B, W, Z)$;
```

b) v is f-special and $g^f(v) \leq g^b(w)$ as well as $g^f(v) \leq g^f(w)$ for all $w \in V_P$.

Proposition 5.11 implies that a b-special or an f-special vertex on P exists. Thus, if a doubling vertex $v \in V_P$ is determined in Line 10, then v is b-special or f-special. Moreover, Proposition 5.11 implies that, if the doubling vertex v determined in Line 10 satisfies $g^b(v) \leq g^f(v)$, then v is b-special and satisfies $g^b(v) \leq \frac{1}{d} - 1$, which means that Case 2a) from the proof of Lemma 5.7 applies. Otherwise, the doubling vertex v determined in Line 10 is f-special and satisfies $g^f(v) \leq \frac{1}{d} - 1$, which means that Case 2b) from the proof of Lemma 5.7 applies.

In the construction in the proof of Lemma 5.7, vertices were identified with their labels. In the implementation, the vertices of G are not renamed. From now on, $L(v)$ is used to refer to the label of a vertex $v \in V$ and $L^{-1}(\ell)$ is used to refer to the vertex which received label $\ell \in [n]$. For example, for a vertex $v \in V$, the vertex whose label comes m steps after the label of v in the numeration is $L^{-1}(L(v)+m)$. The definition of N_m and N_m^{-1} are adjusted to return vertices and to receive vertices and not labels, i. e., for each $v \in V$, let $N_m(v) = L^{-1}(L(v) + m)$ and $N_m^{-1}(v) = L^{-1}(L(v) - m)$.

So, all that is left to do is an analysis of the running time of Algorithm 5.3. We will show that each line can be implemented to run in $\mathcal{O}(n)$ time, which gives a total running time of $\mathcal{O}(n)$. As $\operatorname{diam}^*(G) = \frac{1}{n}|V_P|$, and by Lemma 5.17 and Lemma 5.18, Lines 1-3 take $\mathcal{O}(n)$ time. The algorithm stores the P-labeling in two arrays as discussed before Lemma 5.18. Therefore, the algorithm can determine the label of a vertex v in constant time and also, when given a label, it can determine the corresponding vertex in constant time. To execute Line 4, i. e., to check whether Case 1 applies, the algorithm needs to check if there is a vertex $v \in V_P$ with $N_m(v) \in V_P$. As a vertex $v \in V$ lies in V_P if and only if its path-vertex is v itself, the algorithm can check in constant time whether a vertex $v \in V$ lies in V_P. So, by traversing the list L_P, the algorithm can execute Line 4 in $\mathcal{O}(|V_P|) = \mathcal{O}(n)$ time. Moreover, if executed, this yields also the vertex v that is chosen in Line 5. The sets in Line 6 can be read off the P-labeling in $\mathcal{O}(n)$ time.

From now on, assume that Case 1 does not apply, so Lines 8-22 are executed. To compute the numbers in Line 8, note that a vertex $x \in V_P$ lies in P_v^b if and only if the path-vertex of $N_m(x)$ is v. Indeed, $P_v^b = N_m^{-1}(T_v') \cap V_P = N_m^{-1}(T_v' \cup \{v\}) \cap V_P$ because Case 2 applies when Line 8 is executed. First, the algorithm sets $p^b(v) = 0$ for all $v \in V_P$. Then, it traverses all vertices x in the list L_P and increases $p^b(v)$ by one for the path-vertex v of $N_m(x)$. While doing so, for each $v \in V_P$, the algorithm keeps track of the smallest and the largest vertex $\hat{v} \in T_v'$ such that there is an $x \in V_P$ with $N_m(x) = \hat{v}$. For each $v \in V_P$, if such vertices, say $\hat{v}_s$ and $\hat{v}_\ell$ with $L(\hat{v}_s) \leq L(\hat{v}_\ell)$, exist, then $N_m^{-1}(\hat{v}_s) = v^b$ and $N_m^{-1}(\hat{v}_\ell) = v_\ell^b$ and $h^b(v) = \hat{v}_\ell - \hat{v}_s + 1 - p^b(v)$, since each vertex between v^b and v_ℓ^b lies in $H_v^b \cup P_v^b$. For each $v \in V_P$, if such vertices do not exist, then v is not b-special and the algorithm sets $h^b(v) = 0$. Using the same ideas, the values $p^f(v)$ and $h^f(v)$ for every $v \in V_P$ can be computed. All in all, the computation of the values in Line 8 takes $\mathcal{O}(|V_P|) = \mathcal{O}(n)$ time. Since $v \in V_P$ is b-special if and only if $p^b(v) \neq 0$ and $v \in V_P$ is f-special if and only if $p^f(v) \neq 0$, the algorithm can now determine in constant time whether a vertex $v \in V_P$ is b-special or f-special. Therefore, Line 9 and Line 10 together take $\mathcal{O}(|V_P|) = \mathcal{O}(n)$ time. From now on, denote by v the vertex determined in Line 10.

Assume that Lines 12-14 are executed. Then, the vertex v is b-special as argued above. Using the vertices $\hat{v}_s$ and $\hat{v}_\ell$, that were computed when determining $p^b(v)$ and that must exist since v is b-special, the algorithm can compute v^b and v_ℓ^b in constant time. Thus, the set Z in Line 12, which is the set of vertices between v^b and v_ℓ^b, can be read off the P-labeling in $\mathcal{O}(n)$ time. The vertex $\tilde{v}$ in Line 13 can be obtained from the list L_P and, hence, Line 13 takes $\mathcal{O}(n)$ time. Line 14 can be executed in $\mathcal{O}(n)$ time by using the P-labeling. Similarly to Lines 12-14, Lines 16-18 can be executed in $\mathcal{O}(n)$ time. Line 20 takes constant time as the algorithm can keep track of the size of B_1 while computing it. The subgraph $G[T_v']$ can be computed in $\mathcal{O}(n)$ time by a depth-first traversal of G that is started at v, ignores the neighbors of v that are in V_P, and in the end deletes v. According to Lemma 4.5b), Line 21 then takes $\mathcal{O}(n + |T_v'|) = \mathcal{O}(n)$ time. As B_1 and B_2 are stored as lists and are disjoint as argued in the construction, Line 22 takes $\mathcal{O}(1)$ time. Finally, Line 24 takes $\mathcal{O}(n)$ time by Proposition 2.20. □

This completes the description of the subroutines needed for the algorithm in Theorem 5.14.

Proof of Theorem 5.14. Let G be a forest on n vertices and fix an arbitrary $m \in [n]$. Recall that Algorithm 5.1, which was presented in the proof of Theorem 5.12, can be applied to compute an m-cut in G. So, it suffices to analyze the running time of Algorithm 5.1. As in the proof of Theorem 5.12, let s^* be the number of executions of the while loop and, for $s \in [s^*]$, denote by n_s the number of vertices of the graph G after the sth execution of the while loop. Define $n_0 = n$. Fix an $s \in [s^*]$, and consider the sth execution of the while loop. Then, the application of Lemma 5.13 in Line 3 takes $\mathcal{O}(n_{s-1})$ time by Lemma 5.19 and a list of the vertices in $\tilde{B}$ as well as a list of the vertices in $\tilde{Z}$ are computed. Furthermore, the union in Line 4 is disjoint by (ii) from the proof of Theorem 5.12 and, hence, takes constant time when the set B is stored as a list. With the list of the vertices in $\tilde{Z}$, the update of G in Line 5 takes $\mathcal{O}(n_{s-1})$

time by Corollary 2.23. Therefore, the s^{th} execution of the while loop takes $\mathcal{O}(n_{s-1})$ time. Using (iv) from the proof of Theorem 5.12, i. e., $n_s \le \frac{1}{2^s} n$ for every $s \in [s^*] \cup \{0\}$, this results in a total running time of

$$\mathcal{O}(n) + \sum_{s=1}^{s^*} \mathcal{O}(n_{s-1}) \;=\; \mathcal{O}\left(n + \sum_{s=1}^{s^*} \frac{1}{2^{s-1}} n\right) \;=\; \mathcal{O}(n). \qquad \square$$

5.3 Extension to Tree-Like Graphs

In this section, the inequality relating the minimum bisection width and the diameter of a tree in Theorem 5.6 and its improved version in Theorem 5.12 are extended to tree-like graphs. The aim is to prove Theorem 1.9, which was introduced in Section 1.2.3. There, a graph $G = (V, E)$ on n vertices and a tree decomposition $(T, \mathcal{X})$ with $\mathcal{X} = (X^i)_{i \in V(T)}$ of G are considered. In the general case, we do not work with the relative diameter of a graph but the parameter $r(T, \mathcal{X})$. For a path $P \subseteq T$, the *weight* of P with respect to $\mathcal{X}$ is defined as $w_{\mathcal{X}}(P) := \left|\bigcup_{i \in V(P)} X^i\right|$ and the *relative weight* of P with respect to $\mathcal{X}$ is defined as

$$w^*_{\mathcal{X}}(P) \;:=\; \frac{w_{\mathcal{X}}(P)}{\left|\bigcup_{i \in V(T)} X^i\right|} \;=\; \frac{1}{n}\, w_{\mathcal{X}}(P).$$

A path P' that satisfies $w_{\mathcal{X}}(P') \ge w_{\mathcal{X}}(P)$ for all paths P in T is called a *heaviest path* in T with respect to $\mathcal{X}$ and $r(T, \mathcal{X}) := w^*_{\mathcal{X}}(P')$ is called the *relative weight of a heaviest path* in T with respect to $\mathcal{X}$. Observe that the relative weight of a path is at most one and $r(T, \mathcal{X}) \le 1$ holds for every tree decomposition $(T, \mathcal{X})$, which is similar to $\operatorname{diam}^*(\tilde{G}) \le 1$ for every forest $\tilde{G}$. Furthermore, if $(T, \mathcal{X})$ is a path decomposition, then $r(T, \mathcal{X}) = 1$, which is similar to $\operatorname{diam}^*(\tilde{G}) = 1$ if every component of $\tilde{G}$ is a path. The next lemma points out further relations between the relative weight of a heaviest path in a tree decomposition and the relative diameter of a forest.

Lemma 5.20.
Every forest G on n vertices allows a tree decomposition $(T, \mathcal{X})$ of width at most 1 and size $\mathcal{O}(n)$ that satisfies $r(T, \mathcal{X}) \ge \operatorname{diam}^*(G)$.

Proof. Let G be an arbitrary forest on n vertices. Denote by $G_1, \dots, G_\ell$ the components of G. For every $h \in [\ell]$, let $P_h = (V_h, E_h)$ be a longest path in G_h and denote by x_h and y_h the ends of P_h. Note that $\operatorname{diam}^*(G) = \frac{1}{n} \sum_{h \in [\ell]} |V_h|$. Furthermore, for each $h \in [\ell]$, let T_h be the tree obtained from a copy of G_h, where each vertex $v \in V(G_h)$ has been renamed to i_v and each edge $\{v, w\} \in E(G_h)$ has been subdivided by a new vertex called $i_{\{v,w\}}$. Let $T = (V_T, E_T)$ be the tree obtained from the trees $T_1, \dots, T_\ell$ by inserting the edges $\{i_{y_h}, i_{x_{h+1}}\}$ for every $h \in [\ell - 1]$. Apply the same construction to the forest consisting of the paths $P_1, \dots, P_\ell$, denote by P the graph obtained in this way, and note that P is a path in T. To obtain a tree decomposition $(T, \mathcal{X})$ of G with clusters $\mathcal{X} = (X^i)_{i \in V_T}$, for every $i \in V_T$ with $i = i_v$ for some $v \in V(G)$, set $X^i := \{v\}$ and, for every $i \in V_T$ with $i = i_{\{v,w\}}$ for some $v, w \in V(G)$, set $X^i := \{v, w\}$. Clearly, $(T, \mathcal{X})$ satisfies (T1) and (T2). To check that (T3') is satisfied, fix an arbitrary vertex $v \in V(G)$. Then,

$$I_v \;=\; \left\{i \in V_T \colon\; v \in X^i\right\} \;=\; \{i_v\} \cup \left\{i_{\{v,w\}} \colon\; w \in N_G(v)\right\}$$

and, if $|I_v| \ge 2$, then I_v induces a star in T. Hence, (T3') is satisfied and $(T, \mathcal{X})$ is a tree decomposition of G of width at most 1. Furthermore,

$$\|(T, \mathcal{X})\| \;=\; |V_T| + \sum_{i \in V_T} |X^i| \;\le\; 3|V_T| \;\le\; 3(|V(G)| + |E(G)|) \;\le\; 6n$$

and

$$r(T,\mathcal{X}) \;\geq\; w^*_{\mathcal{X}}(P) \;=\; \frac{1}{n}\left|\bigcup_{i\in V(P)} X^i\right| \;\geq\; \frac{1}{n}\sum_{h\in[\ell]}|V_h| \;=\; \operatorname{diam}^*(G). \qquad \square$$

5.3.1 Upper Bound for the Width of Exact Cuts in Tree-Like Graphs

Here, it is shown that a bisection with the properties in Theorem 1.9 exists. First, the existence part of Theorem 1.9 is restated and generalized to m-cuts.

Theorem 5.21 (Theorem 1.9 restated and generalized).
For every graph G on n vertices, every $m \in [n]$, and every tree decomposition $(T,\mathcal{X})$ of G of width at most $t-1$, there is an m-cut (B,W) in G with

$$e_G(B,W) \;\leq\; \frac{t\Delta(G)}{2}\left(\left(\log_2\left(\frac{1}{r(T,\mathcal{X})}\right)\right)^2 + 9\log_2\left(\frac{1}{r(T,\mathcal{X})}\right) + 8\right).$$

As in the proof of Theorem 5.6, the heart of the proof for the previous theorem is a "doubling lemma". There, Lemma 5.7 and its improved version Lemma 5.13 state that either an exact cut of small width exists or a subgraph with at least twice as large diameter as the original graph exists. Here, the relative weight of a path in a tree decomposition is doubled. The next lemma uses the following notation: Consider a tree decomposition $(T,\mathcal{X})$ with $\mathcal{X} = (X^i)_{i\in V(T)}$ and a path $P \subseteq T$ with $P = (i_0, i_1, \ldots, i_\ell)$. The end i_0 is called a *nonredundant* end of P with respect to $\mathcal{X}$ if $X^{i_0} \neq \emptyset$ and, if $\ell > 0$, $X^{i_h} \not\subseteq X^{i_{h-1}}$ for all $h \in [\ell]$. The path P is called *nonredundant* with respect to $\mathcal{X}$ if at least one of its ends i_0 or i_ℓ is a nonredundant end of P with respect to $\mathcal{X}$. Note that, if $(T,\mathcal{X})$ is a nonredundant tree decomposition, then every path $P \subseteq T$ is nonredundant with respect to $\mathcal{X}$.

Lemma 5.22.
For every graph G on n vertices, for every tree decomposition $(T,\mathcal{X})$ of G of width at most $t-1$, for every $m \in [n]$, and for every path $P \subseteq T$ that is nonredundant with respect to $\mathcal{X}$, there is a cut (B,W,Z) in G that satisfies one of the following options

1) *$|B| = m$, $Z = \emptyset$, and $e_G(B,W) \leq 2t\Delta(G)$, or*
2) *$|B| \leq m \leq |B| + |Z|$ with $0 < |Z| \leq \frac{1}{2}n$, $e_G(B,W,Z) \leq t\Delta(G)\log_2\left(\frac{16}{w^*_{\mathcal{X}}(P)}\right)$, and there is a tree decomposition $(T',\mathcal{X}')$ of $G[Z]$ of width at most $t-1$ and a path $P' \subseteq T'$ that is nonredundant with respect to $\mathcal{X}'$ and satisfies $w^*_{\mathcal{X}'}(P') \geq 2w^*_{\mathcal{X}}(P)$.*

In Option 1) in Lemma 5.22, the bound is increased by a factor of $t\Delta(G)$ compared to Lemma 5.13, as Lemma 2.16b) is used now instead of cutting single edges. In Option 2) in Lemma 5.13, cuts resulting from removing a vertex from the tree were applied, which is extended by using Lemma 2.16a). Therefore, an extra factor of t in the bound on $e_G(B,W,Z)$ in Option 2) in Lemma 5.22 appears. Observe that, in Lemma 5.22, when choosing P to be a heaviest path in the tree decomposition $(T,\mathcal{X})$, then the tree decomposition $(T',\mathcal{X}')$ in Option 2) satisfies $r(T',\mathcal{X}') \geq r(T,\mathcal{X})$. As in the previous section, the proof of Lemma 5.22 is long and technical. Therefore, it is postponed to Section 5.3.2 and the proof for Theorem 5.21 is presented first. Before presenting the proof of Theorem 5.21, consider the next proposition about the relative weight of a heaviest path when turning a tree decomposition into a nonredundant one.

Proposition 5.23.
For every tree decomposition $(T,\mathcal{X})$ of a graph G, there is a nonredundant tree decomposition $(T',\mathcal{X}')$ of G such that

- *the width of $(T', \mathcal{X}')$ is the width of $(T, \mathcal{X})$ and*
- *$r(T', \mathcal{X}') \geq r(T, \mathcal{X})$.*

Proof. Let $(T, \mathcal{X})$ with $\mathcal{X} = (X^i)_{i \in V(T)}$ be an arbitrary tree decomposition of some graph G. Denote by $t-1$ the width of $(T, \mathcal{X})$ and let $P \subseteq T$ be a heaviest path in T with respect to $\mathcal{X}$, i. e., $w^*_{\mathcal{X}}(P) = r(T, \mathcal{X})$. Assume that $(T, \mathcal{X})$ is not nonredundant. Then, there exists an edge $\{i, j\} \in E(T)$ with $X^i \subseteq X^j$. Let T' be the tree obtained from T by contracting the edge $\{i, j\}$ and calling the resulting node h. If $\{i, j\} \cap V(P) = \emptyset$, let $P' = P$. If $\{i, j\} \cap V(P) = \{i, j\}$, let P' be the path obtained from P by contracting the edge $\{i, j\}$. If $\{i, j\} \cap V(P) = \{i\}$ or $\{i, j\} \cap V(P) = \{j\}$, then let P' be the path obtained from P by renaming i or j, respectively, to h. Furthermore, let $\mathcal{X}'$ be the collection of clusters obtained from $\mathcal{X}$ by removing the clusters X^i and X^j and inserting the new cluster $X^h = X^j$. Then, $(T', \mathcal{X}')$ is a tree decomposition of G of width $t-1$ and P' is a path in T' with $w_{\mathcal{X}'}(P') \geq w_{\mathcal{X}}(P)$. Thus, $r(T', \mathcal{X}') \geq r(T, \mathcal{X})$. If $(T', \mathcal{X}')$ is not nonredundant, then the same argument can be repeated until a nonredundant tree decomposition of G with the desired properties is obtained. □

The next corollary can be seen as a forerunner of Theorem 5.21 and is a corollary to Lemma 5.22, similarly as Lemma 5.3 follows from Lemma 5.7 as mentioned after Lemma 5.7. Note that Corollary 5.24 treats a special case of Theorem 5.21 for which it presents a bound on the cut width that is better by a constant factor.

Corollary 5.24.
For every graph G on n vertices, every $m \in [n]$, and every tree decomposition $(T, \mathcal{X})$ of G with $r(T, \mathcal{X}) > \frac{1}{2}$, there is an m-cut (B, W) in G with $e_G(B, W) \leq 2t\Delta(G)$, where $t-1$ denotes the width of $(T, \mathcal{X})$.

Proof. Let $G = (V, E)$ be an arbitrary graph and fix some integer $m \in [n]$. Moreover, let $(T, \mathcal{X})$ be a tree decomposition of G with $r(T, \mathcal{X}) > \frac{1}{2}$ and width $t-1$. Due to Proposition 5.23, we may assume that $(T, \mathcal{X})$ is nonredundant. Let P be a heaviest path in T with respect to $\mathcal{X}$ and note that P is nonredundant with respect to $\mathcal{X}$ as $(T, \mathcal{X})$ is nonredundant.

Recall that, for every tree decomposition $(T_H, \mathcal{X}_H)$ of an arbitrary graph H and for every path $P \subseteq T_H$, the relative weight of P_H with respect to $\mathcal{X}_H$ is at most 1. Now, since $w^*_{\mathcal{X}}(P) = r(T, \mathcal{X}) > \frac{1}{2}$, there cannot be a path P' with relative weight at least $2w^*_{\mathcal{X}}(P)$ with respect to $\mathcal{X}'$, where $(T', \mathcal{X}')$ is a tree decomposition of a subgraph $G[Z]$ of G with $Z \neq \emptyset$. Hence, when Lemma 5.22 is applied to G, $(T, \mathcal{X})$, and the path P with size-parameter m, Option 2) cannot occur. Therefore, Option 1) implies that a cut with the desired properties exists. □

Proof of Theorem 5.21. Let $G = (V, E)$ be an arbitrary graph on n vertices and fix some integer $m \in [n]$. Let $(T, \mathcal{X})$ be an arbitrary tree decomposition of G of width at most $t-1$. We will show that Algorithm 5.4 produces an m-cut in G with the desired properties.

First of all, the while loop in Lines 5-9 eventually terminates as the graph G shrinks in each round due to $|\tilde{Z}| < |V(G)|$. The set B is initialized as the empty set in Line 4 and then only modified in Line 7. As the set $\tilde{B}$ computed in Line 6 contains at most $m - |B|$ vertices, the returned set B will have size m as required. Before starting to analyze the width of the cut produced by Algorithm 5.4, some invariants are stated and it is shown that Algorithm 5.4 can be carried out.

Let s^* be the number of executions of the while loop. Define $G_0 = G$, let $(T_0, \mathcal{X}_0)$ be the tree decomposition $(T, \mathcal{X})$ after Line 1 has been executed, let P_0 be the path P computed in Line 3, and set $B_0 = \emptyset$. Observe that these are the states of the corresponding variables before executing the

Algorithm 5.4: Computes an m-cut.

```
   Input: graph $G = (V, E)$ on $n$ vertices, integer $m \in [n]$, and a tree decomposition $(T, \mathcal{X})$ of $G$.
   Output: an $m$-cut $(B, W)$ in $G$.
1  Transform $(T, \mathcal{X})$ into a nonredundant tree decomposition of $G$ as in Proposition 5.23;
2  Let $G_0$ be a copy of $G$;
3  Compute a heaviest path $P$ in $T$ with respect to $\mathcal{X}$;
4  $B \leftarrow \emptyset$;
5  While $|B| < m$ do
6      Apply Lemma 5.22 to the graph $G$, the tree decomposition $(T, \mathcal{X})$, and the path $P$, with
         size-parameter $\tilde{m} = m - |B|$ to obtain a partition $(\tilde{B}, \tilde{W}, \tilde{Z})$ of $V(G)$ as described there;
7      $B \leftarrow B \,\dot\cup\, \tilde{B}$,    $G \leftarrow G[\tilde{Z}]$;
8      If Option 2) occurred during the application of Lemma 5.22, update $(T, \mathcal{X})$ to a tree
         decomposition $(T', \mathcal{X}')$ of $G$ without increasing its width and update $P$ to a path $P' \subseteq T'$ that is
         nonredundant with respect to $\mathcal{X}'$ and satisfies $w^*_{\mathcal{X}'}(P') \geq 2w^*_{\mathcal{X}}(P)$;
9  Endw
10 Return $(B, V(G_0) \setminus B)$;
```

while loop for the first time. For each $s \in [s^*]$, denote by G_s the graph G, by $(T_s, \mathcal{X}_s)$ the tree decomposition $(T, \mathcal{X})$, by P_s the path P, and by B_s the set B after the s^{th} execution of the while loop. Furthermore, for $s \in [s^*] \cup \{0\}$, let n_s be the number of vertices of G_s and define $w_s := w^*_{\mathcal{X}_s}(P_s)$. For $s \in [s^*]$, denote by $(\tilde{B}_s, \tilde{W}_s, \tilde{Z}_s)$ the partition of the vertex set of G_{s-1} computed in Line 6 during the s^{th} execution of the while loop. At the beginning of each execution of the while loop, the following invariants hold:

(i) $0 < m - |B| \leq |V(G)|$,
(ii) the set B and the vertex set of G are disjoint,
(iii) $(T, \mathcal{X})$ is a tree decomposition of G of width at most $t - 1$, and
(iv) $P \subseteq T$ is a nonredundant path with respect to $\mathcal{X}$.

Furthermore, for every $s \in [s^*]$, the following holds:

(v) $w_s \geq 2w_{s-1}$ for $s \neq s^*$,
(vi) $n_s \leq \frac{1}{2} n_{s-1}$ for $s \neq s^*$, and
(vii) $e_{G_{s-1}}(\tilde{B}_s, \tilde{W}_s, \tilde{Z}_s) \leq t\Delta(G) \log_2\left(\frac{16}{w_{s-1}}\right)$.

By Proposition 5.23, the width of $(T, \mathcal{X})$ does not increase when Line 1 is executed. Furthermore, the path P computed in Line 3 is nonredundant with respect to $\mathcal{X}$ as $(T, \mathcal{X})$ is nonredundant when Line 3 is executed. Thus, (i)-(iv) hold before the first execution of the while loop. Assume that (i)-(iv) hold before the s^{th} execution of the while loop for an arbitrary $s \in [s^*]$. It is now argued that the s^{th} execution of the while loop can be carried out, (i)-(iv) hold before the $(s+1)^{\text{st}}$ execution of the while loop if $s \neq s^*$, and (v)-(vii) hold for s. By (i), (iii), and (iv), it follows that Lemma 5.22 can be applied with size-parameter $\tilde{m} = m - |B|$, i. e., Line 6 can be carried out. If Option 1) occurs in Line 6, then $s = s^*$ and (v)-(vii) are satisfied for s, as $w_{s-1} \leq 1$ implies that $2 \leq \log_2\left(\frac{16}{w_{s-1}}\right)$. If Option 2) in Lemma 5.22 occurs in Line 6, then (v)-(vii) are satisfied for s by Lemma 5.22. In Line 6, the vertex set of the graph G is partitioned into the sets $\tilde{B}_s$, $\tilde{W}_s$, and $\tilde{Z}_s$ with $|\tilde{B}_s| \leq m - |B_{s-1}|$. By (ii), the union in Line 7 is a disjoint union. Hence, if $|\tilde{B}_s| = m - |B_{s-1}|$, then this is the last execution of the while loop and there is nothing more to prove. So, from now on, assume that $|\tilde{B}_s| < m - |B_{s-1}|$. Then, $s < s^*$, Option 2) of Lemma 5.22 must have occurred, and Line 8 is carried out in the s^{th} execution of the while loop. Hence,

G is updated to $G[\tilde{Z}]$ and $\tilde{B}_s$ is added to the set B in Line 7, i. e., $B_s = B_{s-1} \dot{\cup} \tilde{B}_s$. Moreover,

$$|V(G_s)| \;=\; |\tilde{Z}| \;\geq\; \tilde{m} - |\tilde{B}_s| \;=\; m - |B_{s-1}| - |\tilde{B}_s| \;=\; m - |B_s|$$

and $(\tilde{B}_s, \tilde{W}_s, \tilde{Z}_s)$ is a partition of the vertex set of G_{s-1}. Therefore, (i) and (ii) hold after the s^{th} execution of the while loop. Furthermore, (iii) and (iv) are satisfied after the s^{th} execution of the while loop due to the requirements in Line 8. This completes the proof of the invariants (i)-(vii) and shows that the algorithm can be carried out.

Next, the width of the m-cut computed by Algorithm 5.4 is analyzed. First, note that in Line 1 the relative weight of a heaviest path in $(T, \mathcal{X})$ does not decrease by Proposition 5.23 and, thus, $w_0 \geq r(T, \mathcal{X})$. With (v), it follows that

$$w_s \;\geq\; 2^s r(T, \mathcal{X}) \qquad \text{for every } s \in [s^* - 1] \cup \{0\}. \tag{5.18}$$

Therefore,

$$s^* \;\leq\; \log_2\left(\frac{1}{r(T, \mathcal{X})}\right) + 1, \tag{5.19}$$

as otherwise

$$w^*_{\mathcal{X}_{s^*-1}}(P_{s^*-1}) \;=\; w_{s^*-1} \;\overset{(5.18)}{\geq}\; 2^{s^*-1}\, r(T, \mathcal{X}) \;>\; 2^{\log_2\left(\frac{1}{r(T,\mathcal{X})}\right)}\, r(T, \mathcal{X}) \;=\; 1,$$

which is not possible as the relative weight of a path in a tree decomposition is at most 1. Note that, in the previous equation, the path P_{s^*-1} needs to be used as (v) does not apply for $s = s^*$. The number of cut edges in the final cut (B, W) is at most the sum of the edges cut in each execution of the while loop, when the vertex set of the current graph is partitioned in Line 6. By (vii), this implies that

$$\begin{aligned}
e_G(B, W) \;&\leq\; \sum_{s=1}^{s^*} e_{G_{s-1}}(\tilde{B}_s, \tilde{W}_s, \tilde{Z}_s) \;\leq\; t\Delta(G) \sum_{s=1}^{s^*} \log_2\left(\frac{16}{w_{s-1}}\right) \\
&\overset{(5.18)}{\leq}\; t\Delta(G) \sum_{s=1}^{s^*} \left(\log_2\left(\frac{16}{r(T, \mathcal{X})}\right) - (s-1)\right) \\
&\leq\; t\Delta(G) \left(s^* \log_2\left(\frac{16}{r(T, \mathcal{X})}\right) - \frac{1}{2} s^*(s^*-1)\right) \\
&\leq\; t\Delta(G)\; s^* \left(\log_2\left(\frac{16}{r(T, \mathcal{X})}\right) - \frac{1}{2} s^* + \frac{1}{2}\right).
\end{aligned}$$

The last term is a quadratic equation in s^*, whose maximum value is achieved at $s^* = \log_2\left(\frac{16}{r(T,\mathcal{X})}\right) + \frac{1}{2}$, which is larger than the upper bound given in (5.19). Therefore,

$$\begin{aligned}
e_G(B, W) \;&\leq\; t\Delta(G) \left(\log_2\left(\frac{1}{r(T, \mathcal{X})}\right) + 1\right) \cdot \left(\log_2\left(\frac{1}{r(T, \mathcal{X})}\right) + 4 - \frac{1}{2} \log_2\left(\frac{1}{r(T, \mathcal{X})}\right)\right) \\
&\leq\; \frac{t\Delta(G)}{2} \left(\left(\log_2\left(\frac{1}{r(T, \mathcal{X})}\right)\right)^2 + 9 \log_2\left(\frac{1}{r(T, \mathcal{X})}\right) + 8\right). \qquad \square
\end{aligned}$$

Proof of Lemma 5.7	Proof of Lemma 5.22	
forest $G=(V,E)$ on n vertices, assume that G is a tree,	graph $G=(V,E)$ on n vertices,	tree decomposition $(T,\mathcal{X})$ of G of width at most $t-1$, $T=(V_T,E_T)$, $\mathcal{X}=(X^i)_{i\in V_T}$,
$P=(V_P,E_P)$ longest path in G, ends x_0,y_0,	$R=\bigcup_{i\in V_P} X^i$,	$P=(V_P,E_P)$ nonredundant path in T with respect to $\mathcal{X}$, ends i_0,j_0,
$d=\operatorname{diam}^*(G)$, $\lvert V_P\rvert = dn$,	$\lvert R\rvert = rn$,	$r=w^*_{\mathcal{X}}(P)$,
$T_v=$ components of $G-E_P$, $v\in V_P$, $T'_v=V(T_v)\setminus\{v\}$,	$R_i=\{x\in X^i\colon\ i \text{ is the path-node of } x\}$, $S_i=\bigcup_{j\in V(T_i)} X^j\setminus R$,	$T_i=$ components of $T-E_P$, $i\in V_P$,
label the vertices of G, $N_m(v)=v+m$, $N_m^{-1}(v)=v-m$,	label the vertices of G, $N_m(v)=v+m$, $N_m^{-1}(v)=v-m$,	no labeling of the nodes of T,
$v\in V_P$ b-special (f-special) if $\exists\ x\in T'_v$ with $N_m^{-1}(x)\in V_P$ $(N_m(x)\in V_P)$,		$i\in V_P$ b-special (f-special) if $\exists\ x\in S_i$ with $N_m^{-1}(x)\in R$ $(N_m(x)\in R)$,
$P_v^b=N_m^{-1}(T'_v)\cap V_P$, $\{P_v^b\colon\ v\in V_P \text{ b-special}\}$ is a partition of V_P, similarly for $P_v^f=N_m(T'_v)\cap V_P$,	$U_i^b=N_m^{-1}(S_i)\cap R$, $\{U_i^b\colon\ i\in V_P \text{ b-special}\}$ is a partition of R, similarly for $U_i^f=N_m(S_i)\cap R$,	$P_i^b=\{j\in V_P\colon\ R_j\subseteq U_i^b\}$, $\{P_i^b\colon\ i\in V_P \text{ b-special}\}$ is a partition of V_P, similarly for $P_i^f=\left\{j\in V_P\colon\ R_j\subseteq U_i^f\right\}$,
$v^b=N_m^{-1}(x)$ $(v^f=N_m(x))$ for the smallest $x\in T'_v$ with $N_m^{-1}(x)\in V_P$ $(N_m(x)\in V_P)$,		i^b $(i^f)=$ path-node of $N_m^{-1}(x)$ $(N_m(x))$ for the smallest $x\in S_i$ with $N_m^{-1}(x)\in R$ $(N_m(x)\in R)$,
$H_v^b=\bigcup_{x\in P_v^b\setminus\{v^b\}} T'_x$, $H_v^f=\bigcup_{x\in P_v^f\setminus\{v^f\}} T'_x$,	$H_i^b=\bigcup_{j\in P_i^b\setminus\{i^b\}} S_j$, $H_i^f=\bigcup_{j\in P_i^f\setminus\{i^f\}} S_j$,	
$\exists$ b-special v with $\lvert T'_v\rvert+\lvert H_v^b\rvert\le\left(\frac{1}{d}-1\right)\lvert P_v^b\rvert$ or f-special v with $\lvert T'_v\rvert+\lvert H_v^f\rvert\le\left(\frac{1}{d}-1\right)\lvert P_v^f\rvert$,		$\exists$ b-special i with $\lvert S_i\rvert+\lvert H_i^b\rvert\le\left(\frac{1}{r}-1\right)\lvert U_i^b\rvert$ or f-special i with $\lvert S_i\rvert+\lvert H_i^f\rvert\le\left(\frac{1}{r}-1\right)\lvert U_i^f\rvert$.
$Z=H_v^b\,\dot\cup\,P_v^b$ or $Z=H_v^f\,\dot\cup\,P_v^f$.	$Z=H_i^b\,\dot\cup\,U_i^b$ or $Z=H_i^f\,\dot\cup\,U_i^f$.	

Table 5.2: Overview on the notation used in the proofs of Lemma 5.7 and Lemma 5.22. The left column refers to the former one; the middle and right column both refer to the latter one.

5.3.2 Proof of the Doubling Lemma for Tree-Like Graphs

The goal of this paragraph is to prove Lemma 5.22. The technique is similar to the one used in Section 5.2.2, but more involved as we have to deal with the graph itself and the considered tree decomposition of the graph. Table 5.2 gives an overview of the notation used in this paragraph and relates it to the notation used in Section 5.2.2.

Throughout this paragraph, let $G = (V, E)$ be an arbitrary graph on n vertices and let $(T, \mathcal{X})$ be a tree decomposition of G of width at most $t-1$ with $T = (V_T, E_T)$ and $\mathcal{X} = (X^i)_{i \in V_T}$. Denote by n the number of vertices of G and fix an arbitrary integer $m \in [n]$. Furthermore, let $P = (V_P, E_P)$ be a nonredundant path in T with respect to $\mathcal{X}$ and denote by $r := w^*_{\mathcal{X}}(P)$ the relative weight of P. Furthermore, let i_0 and j_0 be the ends of P such that i_0 is a nonredundant end of P with respect to $\mathcal{X}$.

Notation and Vertex Labeling

The first step of the proof is to define a labeling of the vertices of G, which depends on the path P and is similar to the $\tilde{P}$-labeling used in Section 5.2.2. We refer to both, the labeling from Section 5.2.2 and the labeling defined next as a $\tilde{P}$-labeling because we assume that it is clear from the context whether we work with a forest or a tree decomposition such that $\tilde{P}$ is a subgraph of the decomposition tree. To define a $\tilde{P}$-labeling for an arbitrary graph $\tilde{G}$ with a given tree decomposition, the following notation is needed.

For each node $i \in V_P$, let T_i be the component of $T - E_P$ that contains i and call i the root of T_i. As the root of T_i is in V_P for all i, the set $R := \bigcup_{i \in V_P} X^i$ is called the *set of root vertices with respect to P*. For each vertex $x \in R$, define the *path-node* of x as the node $i \in V_P$ closest to i_0 with $x \in X^i$. Note that, as T is a tree, such a node i is unique. For every $i \in V_P$, let

$$R_i := \left\{x \in X^i \colon\ i \text{ is the path-node of } x\right\}.$$

Note that the path-nodes and the sets R_i depend on the choice of i_0. Furthermore, for each node $i \in V_P$, let S_i be the set of vertices $v \in V \setminus R$ such that $v \in X^j$ for some $j \in V(T_i)$, i. e.,

$$S_i = \bigcup_{j \in V(T_i)} X^j \setminus R.$$

For a node $i \in V_P$ and a vertex $x \in V \setminus R$, the node i is called the *path-node* of x if $x \in S_i$. Part b) of the next proposition shows that S_i and $S_{i'}$ are disjoint for distinct nodes i, i' of P. Therefore, every vertex $x \in V$ has a unique path-node.

For technical reasons, we will refer to the pair $\{i_0, j_0\}$ as an edge of T, even when it is not. For a node $i \in V_P \setminus \{j_0\}$, we say that j is the *node after i on P* if $j \in V_P$ and j is the neighbor of i that comes after i when traversing P from i_0 to j_0. Define i_0 to be the *node after j_0 on P*. If j is the node after i on P, the edge $\{i, j\}$ is called the *edge after i on P*. Moreover, if j is the node after i on P, then the edge $\{i, j\}$ is called the *edge before j on P* and i is the *node before j on P*.

Proposition 5.25.

a) For every $i \in V_P$, the set R_i is nonempty.

b) $\{R_i \colon\ i \in V_P\} \cup \{S_i \colon\ i \in V_P\}$ is a partition of V.

Proof.

a) First, note that $R_{i_0} = X^{i_0}$ and $X^{i_0} \neq \emptyset$ as i_0 is a nonredundant end of P with respect to $\mathcal{X}$. If P consists of only one node, this node is i_0 and there is nothing more to show. So assume that $|V_P| \geq 2$

and that there is an $i \in V_P$ with $R_i = \emptyset$. As $X^{i_0} \neq \emptyset$, it follows that $i \neq i_0$. Let j be the node before i on P. Since $X^i \not\subseteq X^j$ as P is nonredundant with respect to $\mathcal{X}$, there must be some vertex $x \in X^i \setminus X^j$. Property (T3') implies that $T[I_x]$ is connected, where $I_x = \{h \in V_T \colon x \in X^h\}$. Furthermore, $I_x \cap V_P$ is nonempty as $i \in I_x \cap V_P$ and, therefore, $T[I_x \cap V_P]$ is not an empty graph and is connected. Moreover, the node in $I_x \cap V_P$ that is closest to i_0 is the path-node of x. As $i \in I_x \cap V_P$ and $j \notin I_x \cap V_P$, the node i must be the path-node of x, which shows that $x \in R_i$, contradicting the assumption that $R_i = \emptyset$.

b) Let x be an arbitrary vertex in V and define $I_x = \{i \in V_T \colon x \in X^i\}$. The set I_x is nonempty by (T1). If $I_x \cap V_P \neq \emptyset$, then $x \in R_j$ for some $j \in V_P$ and $x \notin R_{j'}$ for all $j' \in V_P \setminus \{j\}$ as well as $x \notin \bigcup_{h \in V_P} S_h$. Otherwise, (T3') implies that $I_x \subseteq V(T_j) \setminus \{j\}$ for some $j \in V_P$ and $I_x \cap V(T_{j'}) = \emptyset$ for all $j' \in V_P \setminus \{j\}$. Thus, $x \in S_j$ and $x \notin S_{j'}$ for all $j' \in V_P \setminus \{j\}$. All in all, every vertex $x \in V$ lies in exactly one of the sets in $\{R_i \colon i \in V_P\} \cup \{S_i \colon i \in V_P\}$. □

When labeling the vertices of G with $\{1, 2, \ldots, n\}$, we say that the vertices in a set $V' \subseteq V$ with $V' \neq \emptyset$ receive *consecutive labels* if there are $\ell, \ell' \in [n]$ with $\ell < \ell'$ such that each vertex in V' receives a label in $\{\ell, \ell+1, \ldots, \ell'\}$, or in $\{1, \ldots, \ell\} \cup \{\ell', \ldots, n\}$ and each vertex with a label in this set is in V'. A *P-labeling* of G with respect to $(T, \mathcal{X})$ is a labeling of the vertices in V with $\{1, 2, \ldots, n\}$ such that the following properties are satisfied:

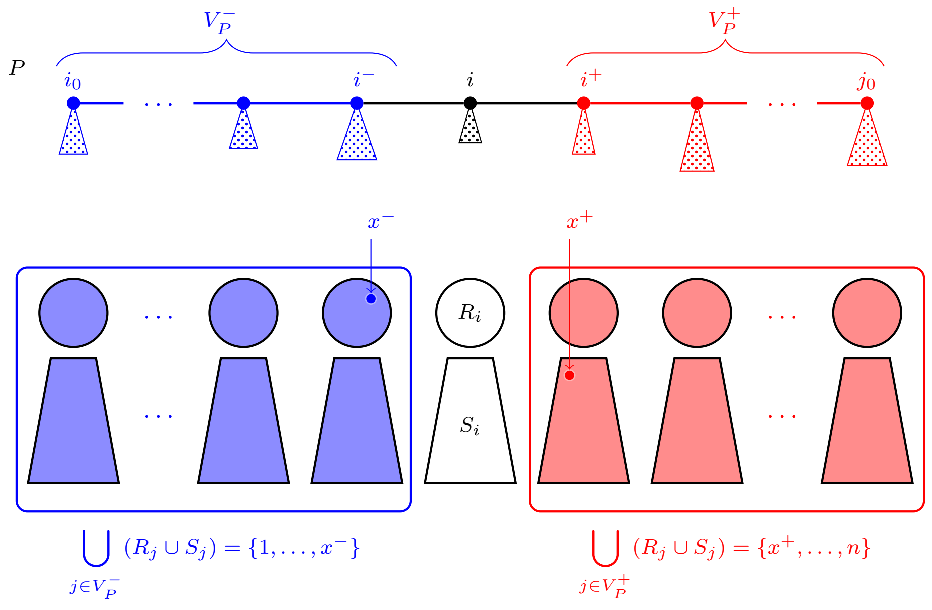
- For each node $i \in V_P$, the vertices in $R_i \cup S_i$ receive consecutive labels and, for each x in S_i and each $y \in R_i$, the label of x is smaller than the label of y.
- For all nodes $i, j \in V_P$ with $i \neq j$, for all vertices $x \in R_i \cup S_i$ and $y \in R_j \cup S_j$, if i_0 is closer to i than j, then the label of x is smaller than the label of y.

Proposition 5.25b) implies that it is always possible to find a P-labeling of the vertices of G. Fix a P-labeling of the vertices of G and identify each vertex with its label. Observe that $1 \in R_{i_0} \cup S_{i_0}$ and $n \in R_{j_0}$. From now on, any number that differs from a label in $[n]$ by a multiple of n is considered to be the same as this label. When referring to labels and vertices, in particular, when comparing them, we always refer to integers in $[n]$. As in Section 5.2.2, for three vertices $a, b, c \in V$ with $a \neq c$, we say that *b is between a and c* if $b = a$, $b = c$, or if starting at a and going along the numeration given by the labeling reaches b before c. If $a = c$, then b is between a and c if $b = a$. For each vertex $x \in V$, the vertex $N_m(x) = x + m$ is called the *m^{th} next vertex of x*. Note that $N_m : V \to V$ is a bijection and, hence, its inverse function N_m^{-1} is well-defined. For a set $Y \subseteq V$, define $N_m(Y) = \{N_m(y) \colon y \in Y\}$.

The next two propositions show how the P-labeling is used to find certain cuts of small width in G. Figure 5.11 visualizes the notation used in the next proposition. In this figure and in all following figures, the path P is drawn on the top and its nodes are drawn explicitly. All nodes of T that do not lie in V_P are not drawn explicitly. For each $i \in V_P$, the tree T_i is represented by the triangle attached to the node i and the sets R_i and S_i are represented by a circle and a trapezoid underneath the node i, respectively.

Proposition 5.26.
Let i be an arbitrary node in P, and denote by i^- and i^+ the nodes before and after i on P, respectively. Let x^- be the vertex with the largest label in R_{i^-} and let x^+ be the vertex with the smallest label in $S_{i^+} \cup R_{i^+}$. Moreover, if $i = i_0$, let $V_P^+ = V_P \setminus \{i_0\}$ and, if $i = j_0$, let $V_P^- = V_P \setminus \{j_0\}$, and otherwise let V_P^- and V_P^+ be the node sets of the components of $P - i$ that contain i^- and i^+, respectively. Removing from G the edges in $E_G(i)$ decomposes G into the following disjoint parts

- *an isolated vertex for every $v \in R_i$,*
- *the part $G[S_i]$,*



P
V_P^-
V_P^+
i_0
i^-
i
i^+
j_0
x^-
x^+
R_i
S_i
$\bigcup_{j\in V_P^-}(R_j\cup S_j)=\{1,\dots,x^-\}$
$\bigcup_{j\in V_P^+}(R_j\cup S_j)=\{x^+,\dots,n\}$

Figure 5.11: Visualization of the notation used in Proposition 5.26.

- *if $i\neq i_0$, the subgraph of G induced by $\bigcup_{j\in V_P^-}(R_j\cup S_j)=\{1,\dots,x^-\}$, and*
- *if $i\neq j_0$, the subgraph of G induced by $\bigcup_{j\in V_P^+}(R_j\cup S_j)=\{x^+,\dots,n\}$.*

Observe that the nodes x^- and x^+ in the previous proposition exist as $R_{i^-}\neq\emptyset$ and $R_{i^+}\neq\emptyset$ due to Proposition 5.25a).

Proof of Proposition 5.26. Let i be an arbitrary node of P and define $\tilde{G}=G-E_G(i)$. By the definition of $E_G(i)$, all vertices in X^i are isolated in $\tilde{G}$. Thus, in $\tilde{G}$, each vertex in R_i is isolated. Assume that $i\neq i_0$ and $i\neq j_0$. The following argument is easy to adjust to the case $i=i_0$ and to the case $i=j_0$. Let $k=\deg_T(i)$ and denote by $i_1,\dots,i_k$ the neighbors of i in T. Without loss of generality, assume that i_1 is the node before i on P and i_k is the node after i on P. For $\ell\in[k]$, let V_ℓ^T be the node set of the component of $T-i$ that contains i_ℓ and define

$$V_\ell=\bigcup_{j\in V_\ell^T} X^j\setminus X^i$$

as in Lemma 2.16. Note that all vertices in $V\setminus X^i$ with a label less than or equal to x^- are in the set V_1, all vertices in S_i are in some set V_ℓ with $\ell\in[k-1]\setminus\{1\}$, and all vertices in $V\setminus X^i$ with a label greater than or equal to x^+ are in V_k. Therefore, Lemma 2.16b) implies that there are no edges in $\tilde{G}$ between a vertex in S_i and a vertex not in S_i. Additionally, there is no edge in $\tilde{G}$ joining two vertices in $V\setminus X^i$ such that one has a label less than or equal to x^- and the other one has a label greater than or equal to x^+. □

Proposition 5.27.
For every node i in the path P and every vertex $x\in R_i$, in the graph $G-E_G(i)$, there is no edge between a vertex with a label less than or equal to x and a vertex with a label greater than or equal to x.

Proof. Let i be an arbitrary node in the path P, let x be a vertex in R_i, and define $\tilde{G}:=G-E_G(i)$. Assume there was an edge $\{y^-,y^+\}$ in $\tilde{G}$ with y^- between 1 and x and y^+ between x and n. Clearly,

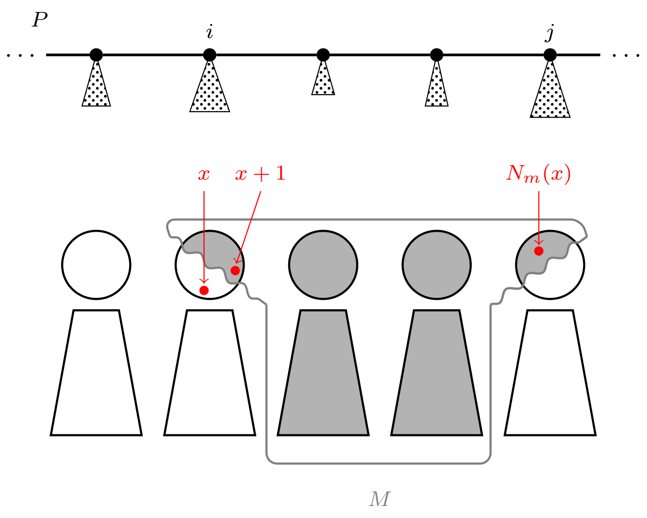


Figure 5.12: The cut (B, W) in Case 1. A set of vertices is colored black or white if its vertices are in the set B or W, respectively. Observe that the black set does not necessarily intersect the set R_i.

$y^- \notin R_i$ and $y^+ \notin R_i$ as all vertices in $R_i \subseteq X^i$ are isolated in $\tilde{G}$. As in Proposition 5.26, denote by i^+ the node after i on P and let x^+ be the smallest vertex in $S_{i^+} \cup R_{i^+}$. Then y^+ must be between x^+ and n and y^- is not between x^+ and n. Hence, Proposition 5.26 shows that $\tilde{G}$ cannot contain the edge $\{y^-, y^+\}$. □

Case 1: There is some $x \in R$ such that $N_m(x) \in R$.

Let B be the set of vertices between $x + 1$ and $N_m(x)$, see Figure 5.12. Let $W = V \setminus B$ and $Z = \emptyset$. Clearly, $|B| = m$. Let $i \in V_P$ and $j \in V_P$ be the path-nodes of x and $N_m(x)$, respectively. Consider the graph $\tilde{G} = G - (E_G(i) \cup E_G(j))$. Then, $e_{\tilde{G}}(B, W) = 0$. Indeed, assume for a contradiction that there is an edge $\{z, z'\} \in E_{\tilde{G}}(B, W)$. Without loss of generality, assume that $z < z'$. Then, either $z \le x < z'$ or $z \le N_m(x) < z'$. However, $G - E_G(i)$ does not contain the edge $\{z, z'\}$ in the first case and $G - E_G(j)$ does not contain the edge $\{z, z'\}$ in the second case by Proposition 5.27. Thus, $\tilde{G}$ does not contain the edge $\{z, z'\}$, which contradicts the assumption. Consequently, every edge that is cut by the cut (B, W) in G is in $E_G(i) \cup E_G(j)$ and

$$e_G(B, W) \;\le\; |E_G(i) \cup E_G(j)| \;\le\; 2t\Delta(G),$$

as desired for Option 1).

Case 2: There is no $x \in R$ with $N_m(x) \in R$.

Then, every $x \in R$ satisfies $N_m(x) \notin R$ and $N_m^{-1}(x) \notin R$. Therefore, $|R| = |N_m(R)| \le |V \setminus R|$, which implies that $|R| \le \frac{1}{2}n$ and, hence,

$$r \;=\; w^*_{\mathcal{X}}(P) \;\le\; \frac{|R|}{|V|} \;\le\; \frac{1}{2}. \tag{5.20}$$

Further Notation and Properties for Case 2

If $i_0 \neq j_0$, and i_0 is not a neighbor of j_0 in T, then denote by T^+ the graph obtained from T by inserting the edge $\{i_0, j_0\}$. Otherwise, let $T^+ = T$. The next proposition presents two easy observations that hold in Case 2.

Proposition 5.28.
In Case 2, the following statements hold:

a) For each $i \in V_P$, there is a node $j \in V_P$ such that $N_m^{-1}(R_i) \subseteq S_j$ and $N_m^{-1}(R_i) \cap S_{j'} = \emptyset$ for every $j' \in V_P \setminus \{j\}$.

b) For each $i \in V_P$, there is a node $j \in V_P$ such that $N_m(R_i) \subseteq S_j$ and $N_m(R_i) \cap S_{j'} = \emptyset$ for every $j' \in V_P \setminus \{j\}$.

Proof.

a) Fix an arbitrary $i \in V_P$. The assumption of Case 2 implies that $N_m^{-1}(R_i) \cap R = \emptyset$ and, therefore, $N_m^{-1}(R_i) \subseteq \bigcup_{j \in V_P} S_j$. Assume for a contradiction that $N_m^{-1}(R_i)$ intersects more than one set S_j with $j \in V_P$. Let x be the smallest vertex in R_i and let j be the path-node of $N_m^{-1}(x)$. Let x' be a vertex in R_i such that the path-node of $N_m^{-1}(x')$ is j' and $j' \neq j$. Since the vertices in $N_m^{-1}(R_i)$ have consecutive labels, all vertices between $N_m^{-1}(x)$ and $N_m^{-1}(x')$ are in $N_m^{-1}(R_i)$. Each vertex in R_j is between $N_m^{-1}(x)$ and $N_m^{-1}(x')$ as the vertices in R_j are labeled right after the vertices in S_j. By Proposition 5.25a) $R_j \neq \emptyset$, which means that there is a vertex $z \in R_j$ with $z \in N_m^{-1}(R_i)$. This contradicts $N_m^{-1}(R_i) \cap R = \emptyset$.

b) Analogous to a). □

A node $i \in V_P$ is called *b-special* if the set S_i contains a vertex x with $N_m^{-1}(x) \in R$. A node $i \in V_P$ is called *f-special* if S_i contains a vertex x with $N_m(x) \in R$. Similarly to Section 5.2.2, we use b as in backward because there is some vertex in S_i such that going m steps backward from x in the numeration gives a vertex in R, and we use f as in forward because there is some vertex in S_i such that going m steps forward in the numeration gives a vertex in R. For every $i \in V_P$, define

$$U_i^b = N_m^{-1}(S_i) \cap R \qquad \text{and} \qquad U_i^f = N_m(S_i) \cap R$$

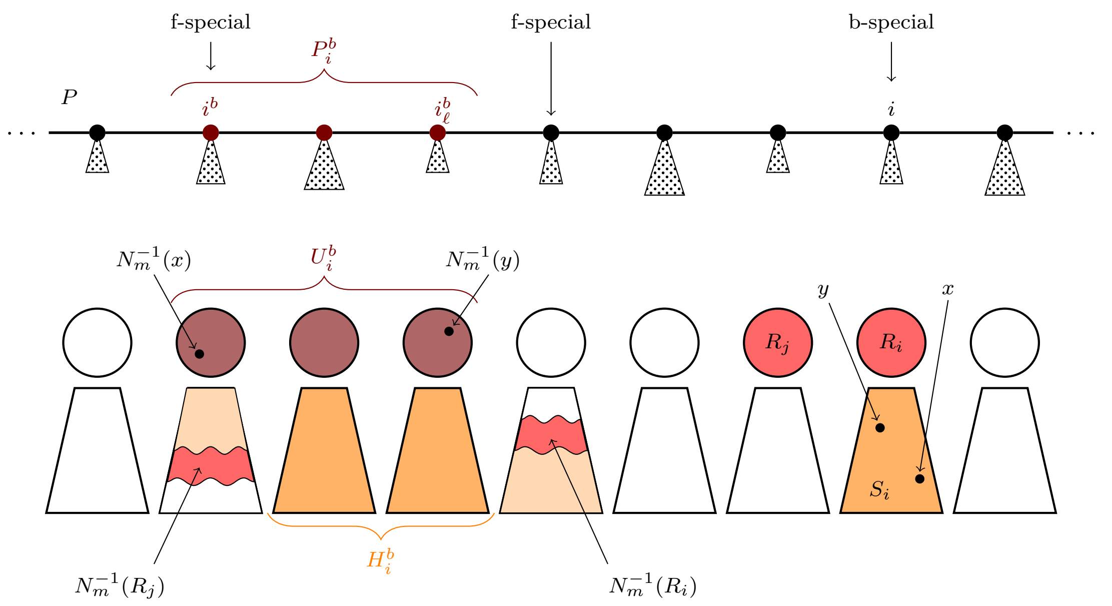
as well as

$$P_i^b = \left\{ j \in V_P \colon\ R_j \subseteq U_i^b \right\} \qquad \text{and} \qquad P_i^f = \left\{ j \in V_P \colon\ R_j \subseteq U_i^f \right\}.$$

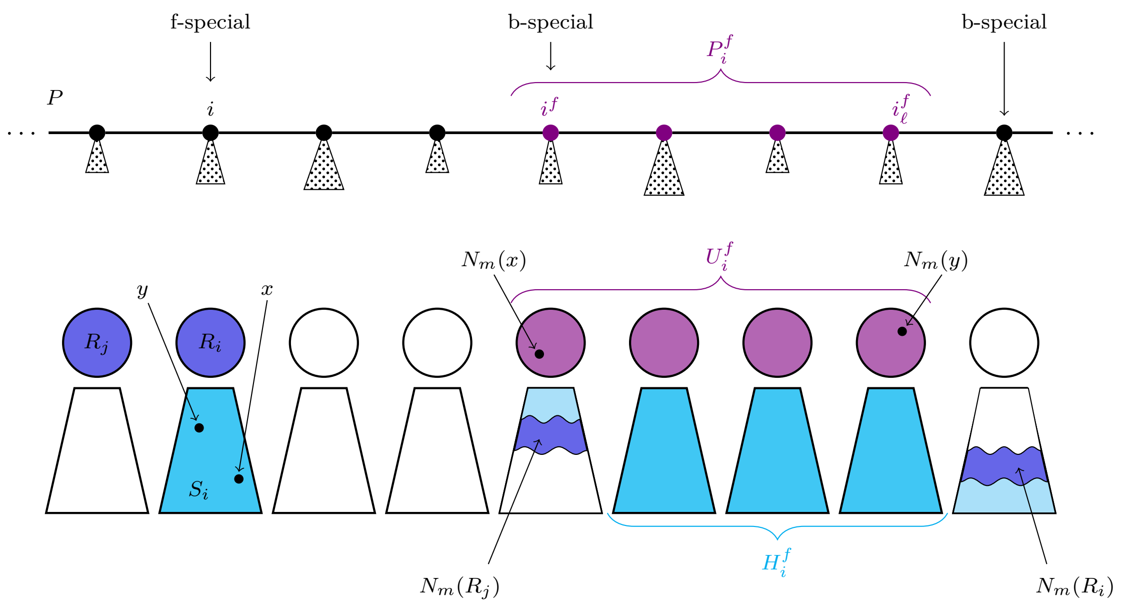
Note that a node $i \in V_P$ is b-special if and only if $U_i^b \neq \emptyset$, and $i \in V_P$ is f-special if and only if $U_i^f \neq \emptyset$. See Figure 5.13 for a visualization of this definition and the following ones. For each b-special $i \in V_P$, let x be the smallest vertex in S_i with $N_m^{-1}(x) \in R$ and let i^b be the path-node of $N_m^{-1}(x)$. Also, for each b-special $i \in V_P$, let y be the largest vertex in S_i with $N_m^{-1}(y) \in R$ and let i_ℓ^b be the path-node of $N_m^{-1}(y)$. We use ℓ as in large, as $R_{i_\ell^b}$ contains $N_m^{-1}(y)$ and y is the largest vertex in S_i with $N_m^{-1}(y) \in R$. Similarly, for each f-special node $i \in V_P$, let x be the smallest vertex in S_i with $N_m(x) \in R$ and let i^f be the path-node of $N_m(x)$. Also, let y be the largest vertex in S_i with $N_m(y) \in R$ and let i_ℓ^f be the path-node of $N_m(y)$. Furthermore, define

$$H_i^b = \bigcup_{j \in P_i^b \setminus \{i^b\}} S_j \qquad \text{and} \qquad H_i^f = \bigcup_{j \in P_i^f \setminus \{i^f\}} S_j$$

for each b-special and f-special $i \in V_P$, respectively, and $H_i^b = \emptyset$ for each $i \in V_P$ that is not b-special as well as $H_i^f = \emptyset$ for each $i \in V_P$ that is not f-special. The next proposition and lemma state some properties about these sets as well as b-special and f-special nodes.



a) A b-special node i and the sets U_i^b, P_i^b, and H_i^b. The sets R_i and R_j, as well as their images under N_m^{-1}, are colored red. The set S_i and its image under N_m^{-1} is colored orange, except for the sets R_h in the image that are subsets of R, which are colored dark red.



b) An f-special node i and the sets U_i^f, P_i^f, and H_i^f. The sets R_i and R_j, as well as their images under N_m, are colored dark blue. The set S_i and its image under N_m is colored light blue, except for the sets R_h in the image that are subsets of R, which are colored purple.

Figure 5.13: Notation used in the proof of Lemma 5.22.

Proposition 5.29.
In Case 2, the following statements hold:

a) For each $i \in V_P$, there is a $j \in V_P$ such that $R_i \subseteq U_j^b$ and $R_i \cap U_{j'}^b = \emptyset$ for every $j' \in V_P \setminus \{j\}$. For each $i \in V_P$, there is a $j \in V_P$ such that $R_i \subseteq U_j^f$ and $R_i \cap U_{j'}^f = \emptyset$ for every $j' \in V_P \setminus \{j\}$.
b) $\{U_i^b \colon i \in V_P\}$ is a partition of R and $\{U_i^f \colon i \in V_P\}$ is a partition of R.
c) $\{P_i^b \colon i \in V_P,\ i$ is b-special$\}$ is a partition of V_P and $\{P_i^f \colon i \in V_P,\ i$ is f-special$\}$ is a partition of V_P.
d) For all $i \in V_P$, if $i \in P_i^b$ then $i = i^b$, and if $i \in P_i^f$ then $i = i^f$.
e) $N_m(U_i^b \cup H_i^b) \subseteq S_i$ for every b-special $i \in V_P$ and $N_m^{-1}(U_i^f \cup H_i^f) \subseteq S_i$ for every f-special $i \in V_P$.

Proof.

a) Fix an arbitrary $i \in V_P$. By Proposition 5.28b), there is a $j \in V_P$ such that $N_m(R_i) \subseteq S_j$ and $N_m(R_i) \cap S_{j'} = \emptyset$ for every $j' \in V_P \setminus \{j\}$. This implies that $R_i \subseteq N_m^{-1}(S_j)$ and $R_i \cap N_m^{-1}(S_{j'}) = \emptyset$ for every $j' \in V_P \setminus \{j\}$. As $R_i \subseteq R$, the first statement follows. The second one follows similarly from Proposition 5.28a).

b) By definition, $U_i^b \subseteq R$ for every $i \in V_P$. Now, Part a) implies that

$$\bigcup_{i \in V_P} U_i^b \ \subseteq\ R \ =\ \bigcup_{i \in V_P} R_i \ \subseteq\ \bigcup_{i \in V_P} U_i^b,$$

which shows that $\bigcup_{i \in V_P} U_i^b = R$. By Proposition 5.25b), the sets S_i with $i \in V_P$ are pairwise disjoint. As N_m^{-1} is a bijection, the sets U_i^b with $i \in V_P$ are also pairwise disjoint. The statement for U_i^f follows similarly.

c) First, note that Part a) implies that $P_i^b \neq \emptyset$ if and only if i is b-special, as well as $P_i^f \neq \emptyset$ if and only if i is f-special. Now, the statement follows immediately from Part a) and Part b) since $R_i \neq \emptyset$ for every $i \in V_P$ by Proposition 5.25a).

d) Let i be an arbitrary node in P and assume that i satisfies $i \in P_i^b$. Recall that $R_j \neq \emptyset$ for all $j \in V_P$ by Proposition 5.25a). Now, i must be b-special, because otherwise $U_i^b = \emptyset$ and also $P_i^b = \emptyset$. Moreover, the node i can only lie in P_i^b if $R_i \subseteq U_i^b$, i. e., $R_i \subseteq N_m^{-1}(S_i)$. Since $R_i \neq \emptyset$ by Proposition 5.25a), the set S_i contains a vertex v such that $N_m^{-1}(v) \in R_i$. Now, for every $w \in S_i$ that is smaller than v, the vertex $N_m^{-1}(w)$ must be in $R_i \cup S_i$. Consequently, $i^b = i$. The second part follows analogously.

e) Consider a b-special node $i \in V_P$. Clearly, $N_m(U_i^b) \subseteq S_i$ by the definition of U_i^b. If $H_i^b = \emptyset$, this is enough. Otherwise, all vertices in H_i^b are between the largest vertex $x' \in R_{i^b}$ and the smallest vertex $y' \in R_{i_\ell^b}$. The definition of i^b and i_ℓ^b together with Proposition 5.28b) imply that $N_m(R_{i^b}) \subseteq S_i$ and $N_m(R_{i_\ell^b}) \subseteq S_i$. Therefore $N_m(x') \in S_i$ and $N_m(y') \in S_i$, implying that $N_m(H_i^b) \subseteq S_i$. The second part for f-special nodes can be shown analogously. □

If P_i^b is not empty, then the nodes in P_i^b induce a path or a cycle in T^+ and due to Proposition 5.29a) the nodes i^b and i_ℓ^b each lie in P_i^b. So, if the nodes in P_i^b induce a path in T^+, then the ends of this path are i^b and i_ℓ^b. Similarly, if $P_i^f \neq \emptyset$, then the nodes in P_i^f induce a path or a cycle in T^+ and, in the former case, the ends of this path are i^f and i_ℓ^f.

Lemma 5.30.
In Case 2, the following statements hold:

a) For every b-special $i \in V_P$, the node i^b is f-special.
b) For every f-special $i \in V_P$, the node i^f is b-special.
c) A node $i \in V_P$ is b-special if and only if there exists an f-special node $j \in V_P$ with $i = j^f$.
d) A node $i \in V_P$ is f-special if and only if there exists a b-special node $j \in V_P$ with $i = j^b$.

Proof.

a) Let $i \in V_P$ be an arbitrary b-special node and denote by j the node before i on P. Proposition 5.28a) implies that there is a node $h \in V_P$ such that $N_m^{-1}(R_j) \subseteq S_h$. The node h must be i^b and, therefore, $N_m^{-1}(R_j) \subseteq S_{i^b}$. By Proposition 5.25a) R_j is nonempty and, thus, S_{i^b} contains a vertex x with $N_m(x) \in R_j$. Consequently, i^b is f-special.

b) Let $i \in V_P$ be an arbitrary f-special node and denote by j the node before i on P. By Proposition 5.28b), there is a node $h \in V_P$ with $N_m(R_j) \subseteq S_h$. The node h must be i^f and $N_m(R_j) \subseteq S_{i^f}$. Due to Proposition 5.25a) the set R_j is nonempty and, hence, S_{i^f} contains a vertex x such that $N_m^{-1}(x) \in R_j$. Therefore, i^f is b-special.

c) First, if there is an f-special node $j \in V_P$ such that $i = j^f$, then Part b) shows that i is b-special. Furthermore,

$$\begin{aligned} \left|\{i \in V_P\colon\ i \text{ is b-special}\}\right| &= \left|\{i^b\colon\ i \in V_P,\ i \text{ is b-special}\}\right| \\ \overset{a)}{\leq} \left|\{j \in V_P\colon\ j \text{ is f-special}\}\right| &= \left|\{j^f\colon\ j \in V_P,\ j \text{ is f-special}\}\right| \\ \overset{b)}{\leq} \left|\{i \in V_P\colon\ i \text{ is b-special}\}\right|, \end{aligned}$$

where the first equality holds because Proposition 5.29c) implies that $i^b \neq j^b$ for two distinct b-special nodes $i, j \in V_P$ and the second equality holds analogously. As all inequalities must be equalities, there are exactly as many b-special nodes $i \in V_P$ as there are f-special nodes $i \in V_P$ and, for every b-special $i \in V_P$, there is an f-special node $j \in V_P$ with $i = j^f$.

d) Analog to Part c). □

Accounting for Case 2

Lemma 5.30c) implies

$$\sum_{\substack{j \in V_P\colon \\ j \text{ is f-special}}} \left|S_{j^f}\right| = \sum_{\substack{i \in V_P\colon \\ i \text{ is b-special}}} |S_i| \tag{5.21}$$

and Lemma 5.30d) implies

$$\sum_{\substack{j \in V_P\colon \\ j \text{ is b-special}}} \left|S_{j^b}\right| = \sum_{\substack{i \in V_P\colon \\ i \text{ is f-special}}} |S_i|\,. \tag{5.22}$$

As $U_i^b \subseteq R$ and $H_i^b \subseteq V \setminus R$ are disjoint, Proposition 5.29e) implies

$$\left|U_i^b\right| + \left|H_i^b\right| \leq \left|S_i\right| \qquad \text{for every b-special } i \in V_P. \tag{5.23}$$

Similarly, for U_i^f and H_i^f, it follows that

$$\left|U_i^f\right| + \left|H_i^f\right| \leq \left|S_i\right| \qquad \text{for every f-special } i \in V_P. \tag{5.24}$$

Recall that $\frac{1}{n}|R| = w^*_{\mathcal{X}}(P) = r$. Proposition 5.25b) guarantees that every $v \in V \setminus R$ is in exactly one set S_j with $j \in V_P$ and Proposition 5.29c) states that $\{P_i^b\colon\ i \in V_P,\ i \text{ is b-special}\}$ is a partition of V_P. Consequently, every vertex $v \in V \setminus R$ is in exactly one set $H_i^b \,\dot\cup\, S_{i^b}$ for some b-special $i \in V_P$ and

$$(1-r)n \;=\; |V \setminus R| \;=\; \sum_{\substack{i \in V_P:\\ i \text{ is b-special}}} \left(|S_{i^b}| + |H_i^b|\right). \tag{5.25}$$

Furthermore, Proposition 5.29b) yields

$$rn \;=\; |R| \;=\; \sum_{\substack{i \in V_P:\\ i \text{ is b-special}}} \left|U_i^b\right|. \tag{5.26}$$

Moreover, Proposition 5.29c) states that $\{P_i^f\colon\ i \in V_P,\ i \text{ is f-special}\}$ is a partition of V_P. Hence, it follows similarly that

$$(1-r)n \;=\; |V \setminus R| \;=\; \sum_{\substack{i \in V_P:\\ i \text{ is f-special}}} \left(\left|S_{i^f}\right| + \left|H_i^f\right|\right) \qquad \text{and} \tag{5.27}$$

$$rn \;=\; |R| \;=\; \sum_{\substack{i \in V_P:\\ i \text{ is f-special}}} \left|U_i^f\right|. \tag{5.28}$$

Now, equations (5.25)-(5.28) imply

$$\sum_{\substack{i \in V_P:\\ i \text{ is b-special}}} \left(|S_{i^b}| + |H_i^b|\right) \;+\; \sum_{\substack{i \in V_P:\\ i \text{ is f-special}}} \left(\left|S_{i^f}\right| + \left|H_i^f\right|\right) \;=\; 2\ |V \setminus R|$$

$$=\; 2\ \frac{1-r}{r}\ |R| \;=\; \frac{1-r}{r} \left(\sum_{\substack{i \in V_P:\\ i \text{ is b-special}}} \left|U_i^b\right| \;+\; \sum_{\substack{i \in V_P:\\ i \text{ is f-special}}} \left|U_i^f\right| \right).$$

Now, the terms in the sums in the first expression are rearranged with (5.21) and (5.22), such that both sums use $|S_i|$ instead of $|S_{i^b}|$ and $|S_{i^f}|$, respectively, which yields

$$\sum_{\substack{i \in V_P:\\ i \text{ is b-special}}} \left(|S_i| + |H_i^b|\right) \;+\; \sum_{\substack{i \in V_P:\\ i \text{ is f-special}}} \left(\left|S_i\right| + \left|H_i^f\right|\right)$$

$$=\; \frac{1-r}{r} \left(\sum_{\substack{i \in V_P:\\ i \text{ is b-special}}} \left|U_i^b\right| \;+\; \sum_{\substack{i \in V_P:\\ i \text{ is f-special}}} \left|U_i^f\right| \right).$$

To deduce the next proposition from the previous equation, note that each sum in the previous equation has at least one summand as there is at least one b-special node $i \in V_P$ and at least one f-special node $i \in V_P$. Indeed, let x be an arbitrary vertex in R, then neither $N_m(x) \in R$ nor $N_m^{-1}(x) \in R$ as otherwise Case 2 would not apply. Let i and j be the path-nodes of $N_m(x)$ and $N_m^{-1}(x)$, respectively. Then i is b-special and j is f-special.

Proposition 5.31.
In Case 2, one of the following must exist:

a) a b-special node $i \in V_P$ such that $|S_i| + |H_i^b| \le \left(\frac{1}{r} - 1\right) |U_i^b|$, or

b) an f-special node $i \in V_P$ such that $|S_i| + |H_i^f| \le \left(\frac{1}{r} - 1\right) |U_i^f|$.

The b-special or f-special node guaranteed by the previous proposition will now be used to construct the cut (B, W, Z) for Option 2).

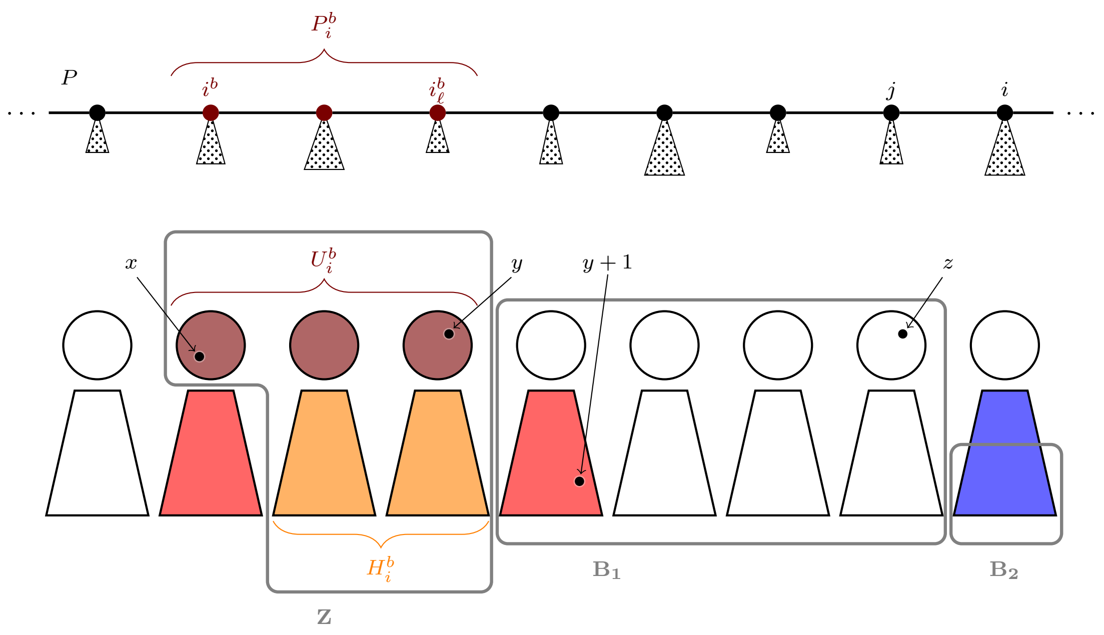


Figure 5.14: Cut (B, W, Z) with $B = B_1 \cup B_2$ in Case 2a). A set S_i is colored blue if the node i is b-special and it is colored red if the node i is f-special. Note that none of the red and blue sets is empty.

The Cut in Case 2a)

Assume there is a b-special node $i \in V_P$ such that $|S_i| + |H_i^b| \leq \left(\frac{1}{r} - 1\right)|U_i^b|$. Replacing $|S_i|$ with (5.23) yields

$$\begin{aligned} \left|U_i^b\right| + 2\left|H_i^b\right| \;&\leq\; \left(\tfrac{1}{r} - 1\right)\left|U_i^b\right| \qquad \Rightarrow \quad 2\left|U_i^b\right| + 2\left|H_i^b\right| \;\leq\; \tfrac{1}{r}\;\left|U_i^b\right| \\ \Rightarrow \quad \left|U_i^b\right| + \left|H_i^b\right| \;&\leq\; \tfrac{1}{2r}\;\left|U_i^b\right| \end{aligned} \tag{5.29}$$

Let $Z := U_i^b \,\dot\cup\, H_i^b$, which satisfies $Z \neq \emptyset$ as i is b-special and, hence, $U_i^b \neq \emptyset$. Furthermore, the sets Z and S_i are disjoint. Indeed, assume there is a vertex $x \in S_i \cap Z$. As $S_i \subseteq V \setminus R$ and $U_i^b \subseteq R$, it follows that $x \in S_i \cap H_i^b$. Since $\{S_h \colon h \in V_P\}$ is a partition of $V \setminus R$ by Proposition 5.25b), the node i must lie in P_i^b and $i \neq i^b$, which contradicts Proposition 5.29d). Thus, the sets S_i and Z are disjoint and (5.23) implies that $2|Z| = 2|U_i^b| + 2|H_i^b| \leq |U_i^b| + |H_i^b| + |S_i| \leq n$ as desired. For the tree decomposition $(T', \mathcal{X}')$ of $G[Z]$ required in Option 2) in Lemma 5.22, consider first the tree decomposition $(T'', \mathcal{X}'')$ induced by $G[Z]$ in $(T, \mathcal{X})$, which is a tree decomposition of $G[Z]$ of width at most $t-1$ by Proposition 2.14. Furthermore, define $P'' := P$. Each vertex in U_i^b is in $R \cap Z$ and contributes to the weight of P'' with respect to $\mathcal{X}''$. Thus, $w^*_{\mathcal{X}''}(P'') \geq \frac{1}{|Z|}|U_i^b|$ and the relative weight of P'' with respect to $\mathcal{X}''$ is at least $2r$, since (5.29) implies $|Z| \leq \frac{1}{2r}|U_i^b|$. Now, let $(T', \mathcal{X}')$ be a nonredundant tree decomposition of $G[Z]$ as in Proposition 5.23, i. e., the width of $(T', \mathcal{X}')$ is the width of $(T'', \mathcal{X}'')$, which is at most $t-1$, and $r(T', \mathcal{X}') \geq r(T'', \mathcal{X}'')$. Let P' be a heaviest path in T' with respect to $\mathcal{X}'$. Then, $w^*_{\mathcal{X}'}(P') = r(T', \mathcal{X}') \geq r(T'', \mathcal{X}'') \geq w^*_{\mathcal{X}''}(P'') \geq 2r$ and P' is nonredundant with respect to $\mathcal{X}'$. Therefore, the set Z and the tree decomposition $(T', \mathcal{X}')$ have the desired properties for Option 2).

Now, the set B for the cut (B, W, Z) in G is defined. See Figure 5.14 for a visualization of the next definitions. Let j be the vertex before i on P. Let y be the largest vertex in $R_{i^b_\ell}$ and let z be the largest vertex in R_j. Both vertices exist as $R_h \neq \emptyset$ for all $h \in V_P$ by Proposition 5.25a). Define

$$B_1 := \{v \in V\colon\ v \text{ is between } y \text{ and } z,\ v \neq y\}.$$

Let $\tilde{m} = m - |B_1|$. As $N_m(y)$ lies in S_i, it follows that $|S_i| \geq \tilde{m} \geq 1$. Let $\tilde{T} = T_i$ and denote by $(\tilde{T}, \tilde{\mathcal{X}})$ the restriction of $(T, \mathcal{X})$ to $\tilde{T}$ and $G[S_i]$. It is easy to see that $(\tilde{T}, \tilde{\mathcal{X}})$ is a tree decomposition of $G[S_i]$, as $I_v = \{h \in V_T\colon\ v \in X^h\} \subseteq V(\tilde{T}) \setminus \{i\}$ for every $v \in S_i$. Let $\tilde{m} := m - |B_1|$ and $c := 2 - \frac{1}{1-r} = \frac{1-2r}{1-r}$ and note that $c \in [0,1)$ due to (5.20). Using the tree decomposition $(\tilde{T}, \tilde{\mathcal{X}})$, Lemma 4.8b) implies that there is a c-approximate $\tilde{m}$-cut (B_2, W_2) in $G[S_i]$ with

$$\begin{aligned} e_{G[S_i]}(B_2, W_2) \ &\leq\ \left\lceil \log_2\left(\frac{1}{1-c}\right)\right\rceil t\Delta(G) \ =\ \left\lceil \log_2\left(\frac{1-r}{r}\right)\right\rceil t\Delta(G) \\ &\leq\ \left(\log_2\left(\frac{2(1-r)}{r}\right)\right) t\Delta(G) \ \leq\ \log_2\left(\frac{2}{r}\right)\cdot t\Delta(G). \end{aligned} \tag{5.30}$$

Define $B = B_1 \cup B_2$. As $B_2 \subseteq S_i$, the set B_2 contains no vertex from B_1 and, thus,

$$\begin{aligned} m - |B| \ &=\ (m - |B_1|) - |B_2| \ \leq\ \tilde{m} - c\tilde{m} \ \leq\ (1-c)|S_i| \\ &\leq\ \frac{r}{1-r}|S_i| \ \leq\ |U_i^b| \ \leq\ |Z|, \end{aligned}$$

where the second to last inequality holds due to the assumption of Case 2a) that $|S_i| + |H_i^b| \leq \left(\frac{1}{r} - 1\right)|U_i^b|$. Thus, $|B| \leq m \leq |B| + |Z|$. The sets B_1 and Z are disjoint by construction. Since $B_2 \subseteq S_i$ and $S_i \cap Z \neq \emptyset$ as shown above, B_2 and Z are disjoint. Consequently, the sets B and Z are disjoint. Let $W := V \setminus (B \dot\cup Z)$.

Next, the width of the cut (B, W, Z) in G is estimated. First, $E_G(S_i, V \setminus S_i) \subseteq E_G(i)$ due to Proposition 5.26. Moreover, $(Z, V \setminus Z)$ cuts only edges in $E_G(i^b) \cup E_G(i^b_\ell)$ due to Proposition 5.26 when considering the nodes i^b and i^b_ℓ. Therefore, $E_G(Z, V \setminus Z) \subseteq E_G(i^b) \cup E_G(i^b_\ell)$. Similarly, the same proposition with considering the nodes i^b_ℓ and i yields $E_G(B_1, V \setminus B_1) \subseteq E_G(i^b_\ell) \cup E_G(i)$. Defining $W_1 = W \setminus W_2$ and recalling that $B_2 \cup W_2 = S_i$, it follows that

$$\begin{aligned} E_G(B, W, Z) \ &\subseteq E_G(Z, B_1, B_2, W_1, W_2) \\ &\subseteq E_G(Z, B_1, S_i, W_1) \cup E_{G[S_i]}(B_2, W_2) \\ &\subseteq E_G(Z, V \setminus Z) \cup E_G(B_1, V \setminus B_1) \cup E_G(S_i, V \setminus S_i) \cup E_{G[S_i]}(B_2, W_2) \\ &\subseteq E_G(i^b) \cup E_G(i^b_\ell) \cup E_G(i) \cup E_{G[S_i]}(B_2, W_2). \end{aligned}$$

Now, (5.30) implies that

$$e_G(B, W, Z) \ \leq\ 3t\Delta(G) + e_{G[S_i]}(B_2, W_2) \ \leq\ \log_2\left(\frac{16}{r}\right) t\Delta(G),$$

because $|E_G(h)| \leq t\Delta(G)$ for every $h \in V_T$.

The Cut in Case 2b)

Assume there is an f-special node $i \in V_P$ such that $|S_i| + |H_i^f| \leq \left(\frac{1}{r} - 1\right)|U_i^f|$. Similarly to Case 2a), using (5.24) yields

$$\left|U_i^f\right| + \left|H_i^f\right| \ \leq\ \tfrac{1}{2r}\left|U_i^f\right|.$$

Now, define $Z = U_i^f \cup H_i^f$ and deduce analogously to Case 2a) that $Z \neq \emptyset$ and $|Z| \leq \frac{1}{2}n$ as S_i and Z are disjoint. Analog to Case 2), a nonredundant tree decomposition $(T', \mathcal{X}')$ of $G[Z]$ of width at most $t-1$ can

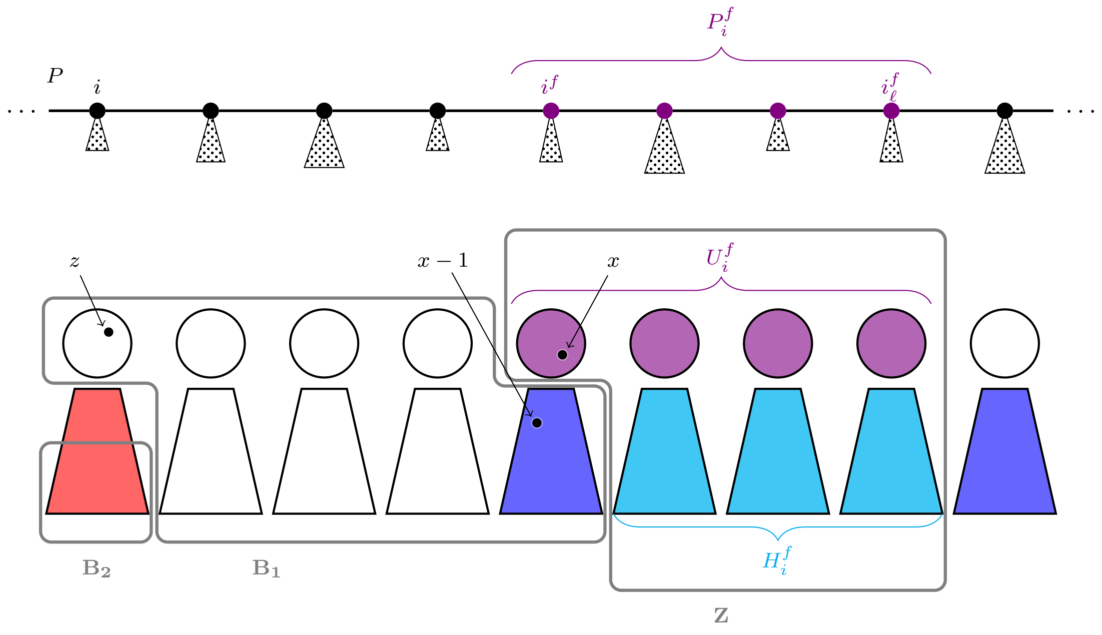


Figure 5.15: Cut (B, W, Z) with $B = B_1 \cup B_2$ in Case 2b). A set S_i is colored dark blue if the node i is b-special and it is colored red if the node i is f-special. Note that none of the red and blue sets are empty.

be constructed from the tree decomposition induced by $G[Z]$ in $(T, \mathcal{X})$, such that a heaviest path $P' \subseteq T'$ with respect to $\mathcal{X}'$ satisfies $w^*_{\mathcal{X}'}(P') \geq 2w^*_{\mathcal{X}}(P)$ and is nonredundant with respect to $\mathcal{X}'$.

Let z be the smallest vertex in R_i and let x be the smallest vertex in R_{i^f}. Define

$$B_1 = \{v \in V\colon\ v \text{ is between } z \text{ and } x,\ v \neq x\},$$

see Figure 5.15. Observe that $B_1 = \emptyset$ if $i = i^f$ or, equivalently, $x = z$. Let $\tilde{m} = m - |B_1|$. As $N_m^{-1}(x)$ is in S_i, it follows that $|S_i| \geq \tilde{m} \geq 1$. As in Case 2a), define $c := 2 - \frac{1}{1-r}$. The restriction $(\tilde{T}, \tilde{\mathcal{X}})$ of $(T, \mathcal{X})$ to $\tilde{T} = T_i$ and $G[S_i]$ is a tree decomposition of $G[S_i]$. Thus, Lemma 4.8b) ensures that there is a c-approximate $\tilde{m}$-cut (B_2, W_2) in $G[S_i]$ with $e_{G[S_i]}(B_2, W_2) \leq \log_2\left(\frac{2}{r}\right) t\Delta(G)$. Defining $B = B_1 \cup B_2$, one can argue that $|B| \leq m \leq |B| + |Z|$ and that B is disjoint from Z. Similarly to Case 2a), it follows with Proposition 5.26 that

$$\begin{aligned}
E_G(Z, V \setminus Z) &\subseteq E_G(i^f) \cup E_G(i_\ell^f),\\
E_G(B_1, V \setminus B_1) &\subseteq E_G(i) \cup E_G(i^f),\\
E_G(S_i, V \setminus S_i) &\subseteq E_G(i).
\end{aligned}$$

For $W = V \setminus (B \cup Z)$, it follows that $e_G(B, W, Z) \leq \log_2\left(\frac{16}{r}\right) t\Delta(G)$.

5.3.3 Computing a Heaviest Path and the Set of P-parameters

Before starting with the details of the main algorithm that computes an exact cut in a tree-like graph, this paragraph focuses on computing a heaviest path, as the construction presented in Sections 5.3.1-5.3.2 relies on a heaviest path in the given tree decomposition. Furthermore, the set of P-parameters is introduced, which consists of several parameters that depend on the path P, for example a P-labeling. Moreover, it is explained how to compute the set of P-parameters.

Algorithm 5.5: Computes a heaviest path in a tree decomposition.

```
Input: integer $n$, tree decomposition $(T, \mathcal{X})$ of a graph $G$ with $V(G) = [n]$.
Output: heaviest path $P \subseteq T$ with respect to $\mathcal{X}$ as an ordered list and its relative weight $r(T, \mathcal{X})$.
1 If $|V(T)| \leq 3$ then
2     Return a list of the nodes in the path $P = T$ and $r(T, \mathcal{X}) = 1$.
3 Endif
4 Root $T$ at an arbitrary node $r$;
5 Compute a leaf $s$ of $T$ with $w(r, s) \geq w(r, s')$ for all leaves $s'$ in $T$;
6 Root $T$ at $s$;
7 Compute a leaf $t$ of $T$ with $w(s, t) \geq w(s, t')$ for all leaves $t'$ in $T$;
8 Return a list of the nodes in the unique $s$,$t$-path in $T$ and $\frac{1}{n} w(s, t)$.
```

The main idea for computing a heaviest path in a tree decomposition is similar to the proof of Lemma 5.17, where a longest path in a tree is computed. Instead of counting the vertices in the considered path, the algorithm now computes the weight of a path in the decomposition tree.

Lemma 5.32.
For every tree decomposition $(T, \mathcal{X})$ of some graph G with $V(G) = [n]$, the relative weight of a heaviest path in T with respect to $\mathcal{X}$ and a heaviest path in T with respect to $\mathcal{X}$ can be computed in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time.

Proof. Let $G = (V, E)$ be a graph with $V = [n]$ for some integer n and let $(T, \mathcal{X})$ be a tree decomposition of G with $\mathcal{X} = (X^i)_{i \in V(T)}$. For two nodes a, b in T, define $W(a, b) := \bigcup_{i \in V(P)} X^i$, where P is the unique a,b-path in T. Furthermore, define $\tilde{W}(a, b) = W(a, b) \setminus X^a$ as well as $w(a, b) = |W(a, b)|$ and $\tilde{w}(a, b) = |\tilde{W}(a, b)|$. Note that, for all $a, b, c \in V(T)$ such that the unique a,c-path in T uses b,

$$w(a, c) \;=\; \left|X^b\right| + \tilde{w}(b, a) + \tilde{w}(b, c) \;=\; w(a, b) + \tilde{w}(b, c),$$

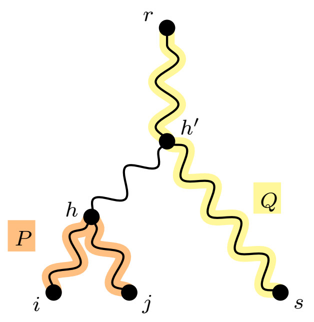
as $W(b, a) \cap W(b, c) = X^b$ by (T3). Apply Algorithm 5.5 to the tree decomposition $(T, \mathcal{X})$ and note that the algorithm only requires $(T, \mathcal{X})$ as input and does not need the underlying graph G. First, it is argued that Algorithm 5.5 indeed computes a heaviest path in $(T, \mathcal{X})$ and afterwards its running time is estimated.

If T contains at most three nodes, then T itself is a path and Algorithm 5.5 is correct. So, from now on, assume that T contains at least four nodes. Denote by r, s, and t the nodes chosen in Algorithm 5.5 when applied to $(T, \mathcal{X})$. Let i and j be two nodes in T such that the unique i,j-path P is a heaviest path in T with respect to $\mathcal{X}$. Without loss of generality, assume that i and j are leaves in T, because otherwise P could be extended, which does not decrease its weight. Denote by h the unique node of P closest to the first root r of T. Let Q be the r,s-path in T.

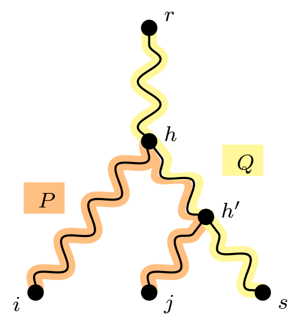
Case 1: P and Q have no common node.

Let h' be the first node on the path from h to r that is in Q, see Figure 5.16a). A node h' with this property always exists as r is in Q. The choice of s implies that $w(r, s) \geq w(r, j)$. As Q and the r,j-path both use the node h', it follows that $w(r, s) = w(r, h') + \tilde{w}(h', s)$ and $w(r, j) = w(r, h') + \tilde{w}(h', j)$, which yields $\tilde{w}(h', s) \geq \tilde{w}(h', j)$. Let T_P be the subgraph of T induced by the nodes in the i,h'-path and the nodes in the j,h'-path. Clearly, $P \subseteq T_P$ and using that the unique i,s-path in T uses h' yields

$$\begin{aligned} w(i, s) \;&=\; w(i, h') + \tilde{w}(h', s) \;\geq\; w(i, h') + \tilde{w}(h', j) \\ &\geq\; \left| \bigcup_{a \in V(T_P)} X^a \right| \;\geq\; \left| \bigcup_{a \in V(P)} X^a \right| \;=\; w(i, j). \end{aligned}$$



a) Case 1, where P and Q have no common node.



b) Case 2, where P and Q have a common node.

Figure 5.16: Proof of Lemma 5.32.

Therefore, the i,s-path is a heaviest path in T with respect to $\mathcal{X}$.

Case 2: P and Q have a common node.

Denote by P_i and P_j the subpaths of P that connect i with h and j with h, respectively. Let h' be the first node on the path from s to r that is in P, see Figure 5.16b). Without loss of generality, assume that h' is in P_j. The choice of s implies that $w(r,s) \geq w(r,j)$, which yields $\tilde{w}(h',s) \geq \tilde{w}(h',j)$ as Q and the r,j-path both use h' and, hence, $w(r,s) = w(r,h') + \tilde{w}(h',s)$ as well as $w(r,j) = w(r,h') + \tilde{w}(h',j)$. Now,

$$w(i,s) = w(i,h') + \tilde{w}(h',s) \geq w(i,h') + \tilde{w}(h',j) = w(i,j),$$

as h' is on P. Thus, the i,s-path is a heaviest path in T with respect to $\mathcal{X}$.

Consequently, in both cases, T contains a heaviest path with respect to $\mathcal{X}$ that ends in s and, therefore, the s,t-path is a heaviest path in T with respect to $\mathcal{X}$.

Next, it is explained how to implement Algorithm 5.5. Denote by n_T the number of nodes of T. Converting T into an arborescence, re-rooting T, and computing a list of the nodes on the s,t-path once s and t are known can be done in $\mathcal{O}(n_T)$ time by Lemma 2.33. Thus, it suffices to show that the node s can be computed in $\mathcal{O}(\|(T,\mathcal{X})\|)$ time when T is an arborescence with root r. For each leaf i of T, let $s(i) = i$. For each node i of T that is not a leaf, let $s(i)$ be a leaf in the subtree rooted at i such that the weight of the i,$s(i)$-path is maximum among all paths that start in i and use only nodes in the subtree rooted at i. Furthermore, for i in T, define $W(i) := W(i, s(i))$ and $w(i) := |W(i)|$. Then, $s = s(r)$ is the desired node. The algorithm traverses T with a depth-first search starting at r and it computes the arrays N and N' as in Lemma 2.30. Recall that $N[i] = |X^i|$ for all $i \in V(T)$ and $N'[i] = |X^i \cap X^{p(i)}|$ for all $i \in V(T) \setminus \{r\}$, where $p(i)$ denotes the parent of i, as well as that $N[i]$ and $N'[i]$ are computed when i turns gray. In addition to the steps of the depth-first traversal the algorithm does the following, when a node i turns black: If i is a leaf, it sets $w(i) = N[i] = |X^i|$ and $s(i) = i$. If i is not a leaf, then

$$w(i) \;=\; \max_{j \text{ is a child of } i} \left\{ \left|W(j) \cup X^i\right| \right\} \;=\; \max_{j \text{ is a child of } i} \left\{ w(j) + \left|X^i\right| - \left|X^i \cap X^j\right| \right\},$$

where the last equality holds because (T3) implies that $W(j) \cap X^i \subseteq X^j$. Let j^* be a child for which this maximum is achieved. The algorithm computes $w(i) = w(j^*) + N[i] - N'[j^*]$ and sets $s(i) = s(j^*)$. Therefore, the algorithm spends $\mathcal{O}(\deg_T(i)+1)$ time at each node $i \in V_T$ in addition to the time needed for the algorithm in Lemma 2.30. Consequently, all values $w(i)$ and $s(i)$ are computed correctly in $\mathcal{O}(\|(T,\mathcal{X})\|)$ time. □

Besides computing a heaviest path, the main algorithm also needs a procedure to compute a P-labeling. While doing so, this procedure also computes further parameters that depend on the path P. Before formally defining this set of parameters, which is called the set of P-parameters, recall the following definitions from Page 179. Some of them depend on the choice of the end i_0 of P. Consider a graph $G = (V, E)$ on n vertices and a tree decomposition $(T, \mathcal{X})$ with $\mathcal{X} = (X^i)_{i \in V(T)}$ of G. Furthermore, let $P = (V_P, E_P) \subseteq T$ be an arbitrary path in T and denote by i_0 and j_0 the ends of P. The set of root vertices with respect to P is $R := \bigcup_{i \in V_P} X^i$. For each vertex $v \in R$, the path-node of v is the node i that is closest to i_0 among all nodes $i \in V_P$ with $v \in X^i$ and, for every $i \in V_P$, define $R_i := \{v \in X^i \colon\ i \text{ is the path-node of } v\}$. Furthermore, for every $i \in V_P$, the tree T_i is the component of $T - E_P$ that contains the node i and $S_i := \bigcup_{j \in V(T_i)} X^j \setminus R$. For each vertex $v \in V \setminus R$, the unique $i \in V_P$ with $v \in S_i$ is called the path-node of v. Moreover, recall that the sets R_i and S_i with $i \in V_P$ form a partition of V, see also Proposition 5.25b). Assume that $V(G) \subseteq [n_0]$, where $n_0 \geq n$ is an arbitrary integer and not necessarily equal to n. This more general definition allows us to work easily with subgraphs $G' \subseteq G$ on n' vertices for which the assumption $V(G') = [n']$ might not hold.

Definition 5.33.
Let $(T, \mathcal{X})$ be a tree decomposition of a graph G on n vertices with $V(G) \subseteq [n_0]$ for some integer n_0 and let $P \subseteq T$ be a path with ends i_0 and j_0. The *set of P-parameters* for G with respect to $(T, \mathcal{X})$ and the end i_0 consists of the following

- a P-labeling of the vertices of G, stored in two integer arrays A_L and A_V of length n_0 and n, respectively, such that, for $v \in V(G) \subseteq [n_0]$, the entry $A_L[v]$ is the label of vertex v and, for $\ell \in [n]$, the entry $A_V[\ell]$ is the vertex that received label ℓ,
- a binary array A_R of length n_0 such that, for each $v \in V$, the entry $A_R[v]$ is one if and only if $v \in R$,
- an integer array A_P of length n_0 such that, for each $v \in V$, the entry $A_P[v]$ is the path-node of v,
- the trees T_i for all i in P, each stored as an arborescence with root i and including, for each node j in $V(T_i)$, a pointer to the corresponding clusters in $\mathcal{X}$, as well as,
- a list L_P of the nodes on P in the order in which they occur when traversing P from i_0 to j_0 including, for each $i \in V(P)$, a pointer to the root of T_i.

Note that the set of P-parameters depends on the choice of i_0. In the remaining part of this chapter, when considering the set of P-parameters of a nonredundant path, then it is assumed that the set of P-parameters is with respect to an end of P that is nonredundant with respect to the clusters $\mathcal{X}$ of the considered tree decomposition. Observe that the v^{th} entry of the arrays A_L, A_R, and A_P may contain an arbitrary value if $v \in [n_0]$ is not a vertex of G. Later on, in Section 5.3.5, this will be useful, because the algorithm does not need to delete these entries from the arrays A_L, A_R, and A_P when adjusting the set of P-parameters of G to obtain the set of P'-parameters of some subgraph $G' \subseteq G$ for some suitably chosen path P' with $V(P') \subseteq V(P)$. The next proposition explains how to use the set of P-parameters to check in constant time, whether $v \in [n_0]$ is a vertex of G and whether $v \in S_i$ for some $i \in V(P)$.

Proposition 5.34.
If the set of P-parameters for G is provided, then

a) for every integer $v \in [n_0]$, one can determine in $\mathcal{O}(1)$ time whether $v \in V(G)$, and

b) for every vertex $v \in V$ and every node $i \in V(P)$, one can decide in $\mathcal{O}(1)$ time whether $v \in S_i$. Furthermore, for every $v \in V(G) \setminus R$, one can determine the unique node $i \in V(P)$ with $v \in S_i$ in constant time.

Proof.

a) Fix an integer $v \in [n_0]$. Now, v is a vertex of G if and only if one of the n entries in A_V contains v. If $A_L[v] \notin [n]$, then v is clearly not a vertex of G. So assume that $A_L[v] \in [n]$. If v is a vertex of G, then $A_V[A_L[v]] = v$. If v is not a vertex of G, then $A_V[A_L[v]] \neq v$ because another vertex of G received label $A_L[v]$. These ideas lead to an algorithm that decides in $\mathcal{O}(1)$ time whether $v \in V(G)$ when given the set of P-parameters as input.

b) Fix some $v \in V$ and some $i \in V_P$. Note that the path-node of v is i if and only if $v \in R_i \cup S_i$. Hence, v lies in S_i if and only if i is the path-node of v and v is not in R, i. e., $A_R[v] = 0$ and $A_P[v] = i$. Clearly, an algorithm can check this in $\mathcal{O}(1)$ time. For the second part, note that, for every $u \in V(G) \setminus R$, there is indeed a unique node $i \in V(P)$ with $u \in S_i$ by Proposition 5.25b). To determine this node i for a vertex $v \in V(G) \setminus R$, the algorithm only needs to determine the path-node of v, which is stored in $A_P[v]$. □

In Section 5.3.2, where it was shown that a cut with the properties in Lemma 5.22 exists, each vertex was identified with its label. The remaining chapter focuses on computing such a cut efficiently. To do so, it is necessary to distinguish between vertices and their labels, because relabeling all vertices and in particular going through the tree decomposition to relabel the vertices might take too long. In the implementation, unless stated explicitly, the algorithm always keeps the original name of each vertex and the labels only appear in the arrays A_L and A_V in the set of P-parameters. In particular, the arrays A_R and A_P of the set of P-parameters as well as the clusters of the tree decomposition refer to the original vertex names and not their labels. Within the remaining section, when considering a labeling of the vertices, $L(v)$ denotes the label of vertex v and $L^{-1}(\ell)$ denotes the vertex, whose label is ℓ. Furthermore, the definition of $N_m(v)$ is adjusted such that it refers to a vertex: For a vertex v, $N_m(v)$ is the vertex w whose label is obtained by adding m to the label of v, i. e., $N_m(v) = L^{-1}(L(v) + m)$. The definition of $N_m^{-1}(v)$ is modified analogously.

Lemma 5.35.
For every tree decomposition $(T, \mathcal{X})$ of a graph G on n vertices with $V(G) = [n]$ and for every path P in T with an end i_0, the set of P-parameters for G with respect to $(T, \mathcal{X})$ and the end i_0 can be computed in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time. The graph G is not required as input.

Proof. Let $G = (V, E)$ be an arbitrary graph on n vertices with $V = [n]$ and $(T, \mathcal{X})$ a tree decomposition of G with $\mathcal{X} = (X^i)_{i \in V(T)}$. Denote by n_T the number of nodes of T. Let P be an arbitrary path in T and denote by i_0 and j_0 the ends of P. Let R, R_i, S_i, and T_i be as defined above with respect to the path P and its end i_0. Assume that P is given as a list L_P of nodes, which is ordered according to the order in which the nodes appear on P when traversing P from i_0 to j_0. If P is given as a graph, then the algorithm computes such a list L_P, which takes $\mathcal{O}(\|P\|) = \mathcal{O}(n_T)$ time. First, the algorithm turns T into an arborescence rooted at the end i_0 of P. At the same time, the algorithm also computes the copies of the trees T_i. To do so, for each node $i \in V(P)$, the algorithm sets $p(i) = i$ to denote that i is the root of the tree T_i, inserts a pointer to i in the list L_P as required for the set of P-parameters, and, if $i \neq j_0$, it deletes the child that is the node after i on P from the list of children of i. Thus, computing L_P and the collection of trees T_i together takes $\mathcal{O}(n_T)$ time. Next, the algorithm computes the array A_R by initializing it with zeros and then going through each cluster X^i with i in P and setting the entries of A_R that correspond to vertices in X^i to 1, which takes $\mathcal{O}(n + \|(T, \mathcal{X})\|)$ time.

To compute the P-labeling, the algorithm creates two arrays A_L and A_V, that are both of length n and both initialized with zeros, which takes $\mathcal{O}(n)$ time. Then, it traverses the nodes i in the list L_P and, for each of them, it traverses the tree T_i with a depth-first search starting at i. When a node $j \neq i$ in T_i turns black, the algorithm labels all unlabeled vertices in $X^j \setminus R$. When the root i turns black, it labels all unlabeled vertices in X^i. More precisely, labeling means to fill in the corresponding entries of the arrays A_L and A_V. Note that checking whether a vertex is in R takes constant time due to the array A_R. Furthermore, each vertex $v \in V$ is labeled while the algorithm is traversing a copy of the tree T_i, where i is the path-node of v. Therefore, the labeling is indeed a P-labeling and the algorithm can compute the array A_P while computing the P-labeling. Thus, computing the labeling and the array A_P together takes $\mathcal{O}(n + \|(T, \mathcal{X})\|)$ time. Consequently, the entire algorithm runs in $\mathcal{O}(n + n_T + \|(T, \mathcal{X})\|) = \mathcal{O}(\|(T, \mathcal{X})\|)$ time. □

5.3.4 Algorithm for Tree-Like Graphs

The goal of this subsection is to prove that a bisection with the properties in Theorem 5.21 can be computed in $\mathcal{O}(nt)$ time when given a graph G on n vertices and a nonredundant tree decomposition $(T, \mathcal{X})$ of G of width $t-1$. Here, it is assumed that the input tree decomposition is nonredundant, as Proposition 2.32a) then implies that $\|(T, \mathcal{X})\| = \mathcal{O}(nt)$, which does not necessarily hold when $(T, \mathcal{X})$ is not nonredundant. Before starting with the algorithmic details, consider the following algorithmic version of Proposition 5.23, which implies that an arbitrary tree decomposition can be turned into a nonredundant tree decomposition without increasing the bound on the width of the cut in Theorem 5.21.

Proposition 5.36 (algorithmic version of Proposition 5.23).
There is an algorithm that receives an arbitrary tree decomposition $(T, \mathcal{X})$ of some graph G on n vertices with $V(G) = [n]$ as input and computes a nonredundant tree decomposition $(T', \mathcal{X}')$ of G in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time, such that

- *the width of $(T', \mathcal{X}')$ is the width of $(T, \mathcal{X})$,*
- *$\|(T', \mathcal{X}')\| \leq \|(T, \mathcal{X})\|$, and*
- *$r(T', \mathcal{X}') \geq r(T, \mathcal{X})$.*

Proof. Let $(T, \mathcal{X})$ with $\mathcal{X} = (X^i)_{\in V(T)}$ be an arbitrary tree decomposition of width $t-1$ of some graph G with $V(G) = [n]$. Recall that, in the proof of Proposition 5.23, edges $\{i, j\} \in E(T)$ with $X^i \subseteq X^j$ were successively contracted to obtain a tree decomposition $(T', \mathcal{X}')$ whose width is $t-1$ and that satisfies $r(T', \mathcal{X}') \geq r(T, \mathcal{X})$. Observe that, when contracting an edge, one cluster is discarded and the number of nodes of T shrinks. Thus, $\|(T', \mathcal{X}')\| \leq \|(T, \mathcal{X})\|$. Recalling that the algorithm in Proposition 2.32b) contracts precisely these edges to make the input tree decomposition nonredundant yields the desired algorithm. □

Next, the algorithmic version of Theorem 5.21 is presented.

Theorem 5.37 (algorithmic version of Theorem 5.21).
For every graph G on n vertices with $V(G) = [n]$, every integer $m \in [n]$, and every nonredundant tree decomposition $(T, \mathcal{X})$ of G of width at most $t-1$, an m-cut (B, W) in G with

$$e_G(B, W) \;\leq\; \frac{t\Delta(G)}{2}\left(\left(\log_2\left(\frac{1}{r(T, \mathcal{X})}\right)\right)^2 + 9\log_2\left(\frac{1}{r(T, \mathcal{X})}\right) + 8\right)$$

can be computed in $\mathcal{O}(nt)$ time and requires only the tree decomposition $(T, \mathcal{X})$ as input.

Note that the running time of this algorithm is not linear in the input size, i. e., not linear in $\|(T, \mathcal{X})\|$. For example, consider a tree decomposition $(T, \mathcal{X})$ of width t, where t is not a constant and only few clusters in $\mathcal{X}$ have size close to t. When a linear running time is desired, the implementation becomes more involved, which is discussed later in Section 5.3.5. This subsection focuses on the easier implementation, whose key idea is the following algorithmic version of Lemma 5.22.

Lemma 5.38 (algorithmic version of Lemma 5.22).
Let G be a graph on n vertices with $V(G) = [n]$ and $(T, \mathcal{X})$ a tree decomposition of G of width at most $t-1$. Furthermore, let $P \subseteq T$ be a nonredundant path with respect to $\mathcal{X}$. For every integer $m \in [n]$, a cut (B, W, Z) in G that satisfies one of the following options

1) *$|B| = m$, $Z = \emptyset$, and $e_G(B, W) \leq 2t\Delta(G)$, or*
2) *$|B| \leq m \leq |B| + |Z|$ with $0 < |Z| \leq \frac{1}{2}n$, $e_G(B, W, Z) \leq t\Delta(G)\log_2\left(\frac{16}{w^*_{\mathcal{X}}(P)}\right)$, and there is a tree decomposition $(T', \mathcal{X}')$ of $G[Z]$ of width at most $t-1$ and a path $P' \subseteq T'$ that is nonredundant with respect to $\mathcal{X}'$ and satisfies $w^*_{\mathcal{X}'}(P') \geq 2w^*_{\mathcal{X}}(P)$,*

can be computed in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time, when $(T, \mathcal{X})$ and P are provided as input.

Proof. Let $G = (V, E)$ be a graph on n vertices with $V(G) = [n]$ and let $(T, \mathcal{X})$ be a tree decomposition of G of width at most $t-1$. Furthermore, let $P = (V_P, E_P)$ be a nonredundant path in T with respect to $\mathcal{X}$ and fix an integer $m \in [n]$. The existence part of Lemma 5.38 follows immediately from Lemma 5.22. Thus, to prove Lemma 5.38, it suffices to describe a procedure that follows the construction from Lemma 5.22, which is summarized in Algorithm 5.6, and analyze its running time. Note that the algorithm only computes the cut (B, W, Z) and other structures such as $(T', \mathcal{X}')$ or P' in Option 2) do not need to be computed. Here, the same notation as in the proof of Lemma 5.22 is used and A_L, A_V, A_R, A_P, and L_P denote the arrays and the list of the set of P-parameters as defined in Definition 5.33. Recall the definitions of the sets R_i, S_i, U_i^b, U_i^f, H_i^b, and H_i^f, see also Page 183 and Table 5.2 on Page 178. Furthermore, for each b-special $i \in V_P$, denote by $x^b(i)$ and $x^b_\ell(i)$ the vertex with the smallest label and the vertex with the largest label among all vertices $x \in S_i$ with $N_m^{-1}(x) \in R$, respectively. Note that the path-node of $N_m^{-1}(x^b(i))$ is i^b and the path-node of $N_m^{-1}(x^b_\ell(i))$ is i^b_ℓ. Similarly, for each f-special $i \in V_P$, denote by $x^f(i)$ and $x^f_\ell(i)$ the vertex with the smallest label and the vertex with the largest label among all vertices $x \in S_i$ with $N_m(x) \in R$, respectively. Then, the path-nodes of $N_m(x^f(i))$ and $N_m(x^f_\ell(i))$ are i^f and i^f_ℓ, respectively.

Similar to Algorithm 5.3, Algorithm 5.6 uses the following additional definitions

$$g^b(i) := \frac{|S_i| + |H_i^b|}{|U_i^b|} \qquad \text{for every b-special } i \in V_P \text{ and}$$

$$g^f(i) := \frac{|S_i| + |H_i^f|}{|U_i^f|} \qquad \text{for every f-special } i \in V_P$$

as well as $g^b(i) := \infty$ for all $i \in V_P$ that are not b-special and $g^f(i) := \infty$ for all $i \in V_P$ that are not f-special. Furthermore, in Line 10, a node $i \in V_P$ is called a *doubling node* if i satisfies one of the following

a) i is b-special and $g^b(i) \leq g^b(j)$ as well as $g^b(i) \leq g^f(j)$ for all $j \in V_P$, or
b) i is f-special and $g^f(i) \leq g^b(j)$ as well as $g^f(i) \leq g^f(j)$ for all $j \in V_P$.

Note that, if Case 2) of the construction in the proof of Lemma 5.22 occurs, i. e., Lines 5-25 are executed, then Proposition 5.31 implies that every doubling node is b-special or f-special. Furthermore, if a doubling node i satisfies $g^b(i) \leq g^f(i)$, then i is b-special and Case 2a) from the proof of Lemma 5.22 applies, and otherwise i is f-special and Case 2b) from the proof of Lemma 5.22 applies. Therefore, the algorithm returns a cut with the desired properties.

Algorithm 5.6: Computes a cut (B, W, Z) with the properties in Lemma 5.38.

```
Input: tree decomposition $(T, \mathcal{X})$ of a graph $G$ on $n$ vertices with $V(G) = [n]$, nonredundant
       path $P = (V_P, E_P) \subseteq T$ with respect to $\mathcal{X}$, and an integer $m \in [n]$.
Output: cut $(B, W, Z)$ with the properties stated in Lemma 5.38.
Compute the set of $P$-parameters for $G$;
If there is a vertex $v \in R$ with $L^{-1}(L(v) + m) \in R$ then
    Let $v$ be a vertex such that $v \in R$ and $L^{-1}(L(v) + m) \in R$;
    $B \leftarrow \{w\colon\ L(w)$ is between $L(v) + 1$ and $L(v) + m\}$,    $Z \leftarrow \emptyset$;
Else
    Compute the values $s(i) := |S_i|$ for all $i \in V_P$;
    Compute the values $u^b(i) := |U_i^b|$ and $h^b(i) := |H_i^b|$ for all $i \in V_P$, as well as the vertices $x^b(i)$
      and $x_\ell^b(i)$ for all b-special $i \in V_P$;
    Compute the values $u^f(i) := |U_i^f|$ and $h^f(i) := |H_i^f|$ for all $i \in V_P$, as well as the vertices $x^f(i)$
      and $x_\ell^f(i)$ for all f-special $i \in V_P$;
    Compute $g^b(i)$ and $g^f(i)$ for all $i \in V_P$;
    Let $i$ be a doubling node;
    If $g^b(i) \leq g^f(i)$ then
        $Z \leftarrow U_i^b \cup H_i^b$;
        Let $y$ be the largest vertex in $R_{i_\ell^b}$;
        Let $z$ be the largest vertex in $R_j$, where $j$ denotes the node before $i$ on $P$;
        $B_1 \leftarrow \{v \in V\colon\ L(v)$ is between $L(y)$ and $L(z)$, $v \neq y\}$;
    Else
        $Z \leftarrow U_i^f \cup H_i^f$;
        Let $z$ be the smallest vertex in $R_i$;
        Let $y$ be the smallest vertex in $R_{i^f}$;
        $B_1 \leftarrow \{v \in V\colon\ L(v)$ is between $L(z)$ and $L(y)$, $v \neq y\}$;
    Endif
    $r \leftarrow w^*_{\mathcal{X}}(P)$,    $c \leftarrow 2 - \frac{1}{1-r}$,    $\tilde{m} \leftarrow m - |B_1|$;
    Let $(B_2, W_2)$ be a $c$-approximate $\tilde{m}$-cut in $G[S_i]$ with $e_{G[S_i]}(B_2, W_2) \leq \log_2\left(\frac{2}{r}\right) t\Delta(G)$ where $t - 1$
      is the width of $(T, \mathcal{X})$;
    $B \leftarrow B_1 \cup B_2$;
Endif
$W \leftarrow V \setminus (B \cup Z)$;
Return $(B, W, Z)$;
```

As in Algorithm 5.3, in the implementation of Algorithm 5.6, the vertices of the input graph are not identified with their labels. From now on, $L(v)$ is used to refer to the label of a vertex $v \in V$ and $L^{-1}(\ell)$ is used to refer to the vertex which received label $\ell \in [n]$. Again, the definition of N_m and N_m^{-1} are adjusted to return vertices and to receive vertices and not labels, i. e., for each $v \in V$, let $N_m(v) = L^{-1}(L(v) + m)$ and $N_m^{-1}(v) = L^{-1}(L(v) - m)$.

Next, the running time of Algorithm 5.6 is analyzed. For a list L the number of entries of L is denoted by $|L|$. Furthermore, the algorithm stores all sets as unordered lists. By Lemma 5.35, Line 1 takes time proportional to $\|(T, \mathcal{X})\|$ and the arrays A_L, A_R, and A_P computed there each have length n. Line 2 takes time proportional to n, by going through the array A_R once and checking for every v with $A_R[v] = 1$

whether $A_R[v'] = 1$, where $v' = L^{-1}(L(v) + m)$ can be determined easily with the arrays A_L and A_V. If such a vertex exists, this procedure also gives the vertex v for Line 3. Furthermore, the set B in Line 4 can be read off the array A_V in $\mathcal{O}(n)$ time.

To implement Lines 6-8, note that all values computed there can be stored in arrays, indexed with V_T as we may assume that $V_T = [n_T]$ for some integer n_T due to Lemma 2.22. First, the algorithm initializes all these arrays with zeros, which takes $\mathcal{O}(n_T)$ time. Then, to compute $s(i) := |S_i|$ for all $i \in V_T$, note that $v \in S_i$ if and only if the path-node of v is i and $i \notin R$, see also Proposition 5.34b). Thus, the algorithm can go through all vertices $v \in V = [n]$ and, for each $v \notin R$, it increases $s(i)$ by one, where i is the path-node of v. This takes $\mathcal{O}(n)$ time as the algorithm can use A_R to check whether $v \in R$ and it can use A_P to determine the path-node of v. Furthermore $v \in U_i^b$ holds if and only if $v \in R$ and $N_m(v) \in S_i$. The vertex $N_m(v)$ can be determined with A_L and A_V and we already discussed how to determine whether a vertex is in S_i. Hence, the algorithm can go through the array A_R and, for every vertex v with $A_R[v] = 1$, it increases the value of $u^b(i)$ by one, where i is the unique node with $v \in U_i^b$. So, computing all values $u^b(i)$ for $i \in V_P$ together takes $\mathcal{O}(n)$ time. While doing so, for each $i \in V_P$, the algorithm computes the vertices $x^b(i)$ and $x_\ell^b(i)$, if they exist. Note that, for $i \in V_P$, the vertices $x^b(i)$ and $x_\ell^b(i)$ exist if and only if i is b-special, i. e., if and only if $u^b(i) > 0$. Moreover, H_i^b is the set of vertices with labels between the labels of $N_m^{-1}(x^b(i))$ and $N_m^{-1}(x_\ell^b(i))$ that are not in U_i^b. Therefore, $h^b(i) = L(x_\ell^b(i)) - L(x^b(i)) + 1 - u^b(i)$ and all values $h^b(i)$ can be computed in $\mathcal{O}(n_T)$ time. Similar to the computation of the values $u^b(i)$ and $h^b(i)$, the algorithm can compute the values $u^f(i)$ and $h^f(i)$ as well as the vertices $x^f(i)$ and $x_\ell^f(i)$, if they exist, for all $i \in V_P$. In total, Lines 6-8 require $\mathcal{O}(n_T + n)$ time.

Recall that a node $i \in V_P$ is b-special if and only if $u^b(i) > 0$ and a node $i \in V_P$ is f-special if and only if $u^f(i) > 0$. Therefore, the algorithm can now determine in constant time whether a node $i \in V_P$ is b-special and the same holds for being f-special. So, Lines 9-10 together take time proportional to $|L_P|$. To execute Line 12, note that the set $Z = U_i^b \cup H_i^b$ is the set of vertices with labels between the labels of $N_m^{-1}(x^b(i))$ and $N_m^{-1}(x_\ell^b(i))$ and, therefore, it can be read off the array A_V in $\mathcal{O}(n)$ time. Furthermore, the vertex y in Line 13 is identical to $N_m^{-1}(x_\ell^b(i))$ and, hence, Line 13 takes constant time. To determine the vertex z with the largest label among all vertices in R_j, the algorithm first determines the node j with the list L_P in $\mathcal{O}(|L_P|)$ time. Then, it checks for every vertex $v \in [n]$, whether $v \in R$ and whether the path-node of v is j by using the arrays A_R and A_P. While doing so, it keeps track of the vertex with the largest label. Hence, Line 14 takes $\mathcal{O}(|L_P| + n)$ time. Then, the set B_1 in Line 15 can be read off the array A_V in $\mathcal{O}(n)$ time. Consequently, Lines 12-15 take $\mathcal{O}(|L_P| + n)$ time and similarly, Lines 17-20 can be executed in $\mathcal{O}(|L_P| + n)$ time.

Line 22 can be executed in constant time if $w_{\mathcal{X}}^*(P)$ is known and the algorithm keeps track of the size of the set B_1 while creating it. Otherwise, the equation $w_{\mathcal{X}}^*(P) = \frac{1}{n}|R|$ can be used to determine r in $\mathcal{O}(n)$ time as $|R|$ is the number of entries of A_R that are set to one and n is the length of A_R. To apply the approximate cut in Line 23 a tree decomposition $(\tilde{T}, \tilde{\mathcal{X}})$ of $G[S_i]$ is needed. To compute this tree decomposition, note that Proposition 5.34b) says that the algorithm can check whether $v \in V$ lies in S_i in constant time. Hence, the tree decomposition $(\tilde{T}, \tilde{\mathcal{X}})$ that was used in the proof of Lemma 5.22, i.e., the restriction of $(T, \mathcal{X})$ to T_i and $G[S_i]$ can be computed in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time by Proposition 2.31. While doing so, the algorithm keeps track of the vertex $v_s \in S_i$ with the smallest label among all vertices in S_i. The labels of the vertices in S_i are precisely the labels between $L(v_s)$ and $L(v_s) + s(i) - 1$. Therefore, it is easy to set up a bijection between the vertices in S_i and the set $[s(i)]$ in order to rename the vertices in the clusters of $\tilde{\mathcal{X}}$ in $\mathcal{O}(\|(\tilde{T}, \tilde{\mathcal{X}})\|)$ time, which is necessary to apply the algorithm contained in Lemma 4.8 to $G[S_i]$, as it requires that the vertex set of $G[S_i]$ is $[s(i)]$. Then, the application of the algorithm contained in Lemma 4.8 to $G[S_i]$ with the tree decomposition $(\tilde{T}, \tilde{\mathcal{X}})$ takes $\mathcal{O}(\|(\tilde{T}, \tilde{\mathcal{X}})\|)$ time. Observe that it is not necessary to compute the graph $G[S_i]$. After the computation the names of the vertices in the set B_2 need to be converted back to their original names, which takes $\mathcal{O}(n) = \mathcal{O}(\|(T,\mathcal{X})\|)$ additional time. So, Line 23 takes time proportional to $\|(T,\mathcal{X})\|$, as $\|(\tilde{T},\tilde{\mathcal{X}})\| \leq \|(T,\mathcal{X})\|$. Then, Line 24 takes constant time as the sets B_1 and B_2 are disjoint and stored as unordered lists. Finally, Proposition 2.20 implies that Line 26 can be executed in $\mathcal{O}(n)$ time as $V(G) = [n]$.

All in all, the running time is proportional to

$$\underbrace{\|(T,\mathcal{X})\|}_{\text{Line } 1} + \underbrace{n}_{\text{Lines } 2\text{-}4} + \underbrace{n_T + n}_{\text{Lines } 6\text{-}8} + \underbrace{|L_P|}_{\text{Lines } 9\text{-}10}$$
$$+ \underbrace{|L_P| + n}_{\text{Lines } 11\text{-}21} + \underbrace{n}_{\text{Line } 22} + \underbrace{\|(T,\mathcal{X})\|}_{\text{Line } 23} + \underbrace{n}_{\text{Lines } 24\text{-}26} . \tag{5.31}$$

As $n \leq \|(T,\mathcal{X})\|$ and $|L_P| \leq |V_T| \leq \|(T,\mathcal{X})\|$, the running time of Algorithm 5.6 is $\mathcal{O}(\|(T,\mathcal{X})\|)$, as desired. □

Note that, in the previous proof, the tree decomposition $(T,\mathcal{X})$ is only used in Line 1 and Line 23 of Algorithm 5.6. All other computations only rely on the set of P-parameters. Furthermore, the assumption, that the vertex set of the input graph G is $[n]$, was only used for the computation of the set W in Line 26 and when traversing arrays of the set of P-parameters. This will become interesting in Section 5.3.5, when the algorithm contained in Theorem 5.37 is improved to run in linear time. Before doing so, the discussion of the easier implementation is completed by using Lemma 5.38 to derive Theorem 5.37. The construction follows the proof of existence from Section 5.3.1, in particular Algorithm 5.4, which is repeated here.

Algorithm 5.4 (repeated): Computes an m-cut.

```
Input: graph $G = (V,E)$ on $n$ vertices, integer $m \in [n]$, and a tree decomposition $(T,\mathcal{X})$ of $G$.
Output: an $m$-cut $(B,W)$ in $G$.
1  Transform $(T,\mathcal{X})$ into a nonredundant tree decomposition of $G$ as in Proposition 5.23;
2  Let $G_0$ be a copy of $G$;
3  Compute a heaviest path $P$ in $T$ with respect to $\mathcal{X}$;
4  $B \leftarrow \emptyset$;
5  While $|B| < m$ do
6      Apply Lemma 5.22 to the graph $G$, the tree decomposition $(T,\mathcal{X})$, and the path $P$, with
         size-parameter $\tilde{m} = m - |B|$ to obtain a partition $(\tilde{B},\tilde{W},\tilde{Z})$ of $V(G)$ as described there;
7      $B \leftarrow B \dot\cup \tilde{B}$,    $G \leftarrow G[\tilde{Z}]$;
8      If Option 2) occurred during the application of Lemma 5.22, update $(T,\mathcal{X})$ to a tree
         decomposition $(T',\mathcal{X}')$ of $G$ without increasing its width and update $P$ to a path $P' \subseteq T'$ that is
         nonredundant with respect to $\mathcal{X}'$ and satisfies $w^*_{\mathcal{X}'}(P') \geq 2w^*_{\mathcal{X}}(P)$;
9  Endw
10 Return $(B, V(G_0) \setminus B)$;
```

Proof of Theorem 5.37. Let $G_0 = (V_0,E_0)$ be a graph with $V_0 = [n_0]$ and $(T_0,\mathcal{X}_0)$ an arbitrary tree decomposition of G_0. Furthermore, fix an integer $m_0 = m \in [n_0]$. To compute an m-cut in G_0 we follow the ideas presented in the proof of Theorem 5.21, in particular Algorithm 5.4. Note that this implies that the algorithm finishes and returns an m-cut with the desired properties. The implementation described here will always work with the tree decomposition or the set of P-parameters. The graph G_0 is not needed as input and the graph G is only used to present the description in a nicer way, as it clarifies the graph corresponding to the tree decomposition. However, the implementation requires that the vertex set of G_0 is $[n_0]$.

In order to apply Lemma 5.22 in Line 6, or its algorithmic version, Lemma 5.38, the vertex set of the considered graph G needs to be $[n]$, where n denotes the number of vertices of G. To ensure this, the algorithm renames the vertices of G whenever G is modified in Line 7. To do so, the algorithm sets up a bijection from $[n]$ to $V(G)$ and updates it as in Lemma 2.21. Then, when updating the set B in Line 7, the algorithm converts back the vertex names that refer to the current subgraph to their original vertex names referring to the input graph.

Furthermore, to execute Line 8 the algorithm also needs to compute a path P' and a tree decomposition $(T', \mathcal{X}')$ with the properties in Option 2) in Lemma 5.22, when this option occurs. Note that this is not done by the algorithm contained in Lemma 5.38. Consider an execution of the while loop of Algorithm 5.4, where Option 2) of Lemma 5.22 occurs. In Line 6, the procedure contained in Lemma 5.38 is applied to a tree decomposition $(T, \mathcal{X})$ of a graph G on n vertices and a nonredundant path $P \subseteq T$ with respect to $\mathcal{X}$. This yields lists of the vertices in each set of the cut $(\tilde{B}, \tilde{W}, \tilde{Z})$ in G, which can be used to update the bijection as in Lemma 2.21d). After doing so, the arrays storing the bijection of the vertex names enable the algorithm to determine whether a vertex $v \in V(G_0)$ is in $\tilde{Z}$ in constant time by Lemma 2.21c). Therefore, the algorithm can compute a tree decomposition $(T', \mathcal{X}')$ of $G' := G[\tilde{Z}]$ in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time by restricting $(T, \mathcal{X})$ to T and G'. While doing so, it can also adjust the vertex names in the clusters in $\mathcal{X}'$ such that they refer to the vertices in the current graph G. Next, the algorithm turns $(T', \mathcal{X}')$ into a nonredundant tree decomposition as in Proposition 5.23 and computes a heaviest path P' in $(T', \mathcal{X}')$. Observe that P' is nonredundant with respect to $\mathcal{X}'$ as well as that $(T', \mathcal{X}')$ and P' coincide with the tree decomposition and the path used for Option 2) in the proof of Lemma 5.22. Furthermore, everything is prepared for the next execution of the while-loop and in particular the application of Lemma 5.38, the algorithmic version of Lemma 5.22, in Line 6, i. e., the vertex set of the current graph G is $[n]$, where n denotes the number of vertices of G, $(T, \mathcal{X})$ is a nonredundant tree decomposition of G and P is a nonredundant path in T with respect to $\mathcal{X}$ in the beginning of the next execution of the while loop.

To complete the proof, only the analysis of the running time of this implementation is missing. Denote by $t-1$ the width of the input tree decomposition $(T_0, \mathcal{X}_0)$. Recall that, in Theorem 5.37, only nonredundant input tree decompositions are considered and, hence, nothing has to be done in Line 1. The remaining preprocessing in Lines 2-4 of Algorithm 5.4 takes time proportional to $\|(T_0, \mathcal{X}_0)\|$ by Lemma 5.32. Let s^* be the number of executions of the while loop and recall that, in the proof of Theorem 5.21, it was argued that s^* is finite. For $s \in [s^*-1]$, denote by $(T_s, \mathcal{X}_s)$ the tree decomposition $(T, \mathcal{X})$ after the s^{th} execution of the while loop and, for $s \in \{0\} \cup [s^*-1]$, denote by n_s the number of vertices of the corresponding (implicit) graph. Fix $s \in [s^*]$ and consider the s^{th} execution of the while loop. Line 6 takes time proportional to $\|(T_{s-1}, \mathcal{X}_{s-1})\|$ by Lemma 5.38. Updating the set B, including the conversion of the vertex names as well as updating the bijection for the vertex names of the implicit graph G in Line 7 takes $\mathcal{O}(n_{s-1})$ time by Lemma 2.21b) and d). If the algorithm has to do something in Line 8, then, as described above, it first computes a restriction of the tree decomposition $(T_{s-1}, \mathcal{X}_{s-1})$, then renames the vertices in the clusters in $\mathcal{X}$, and then modifies the resulting tree decomposition to be nonredundant, which together takes time proportional to $\|(T_{s-1}, \mathcal{X}_{s-1})\|$ by Proposition 2.31 and Proposition 5.36. Computing the path P' takes time proportional to $\|(T_s, \mathcal{X}_s)\| \leq \|(T_{s-1}, \mathcal{X}_{s-1})\|$ by Lemma 5.32. The width of $(T_{s-1}, \mathcal{X}_{s-1})$ is at most $t-1$ and, hence, Proposition 2.32a) implies that $\|(T_{s-1}, \mathcal{X}_{s-1})\| \leq \mathcal{O}(n_{s-1}t)$. Consequently, the s^{th} execution of the while loop takes $\mathcal{O}(n_{s-1}t)$ time. To derive the running time of the entire procedure, recall that invariant (vi) in the proof of Theorem 5.21 implies that $n_s \leq \frac{1}{2^s} n_0$ for all $s \in \{0\} \cup [s^*-1]$.

Thus, the total running time is

$$\mathcal{O}\left(\|(T_0, \mathcal{X}_0)\| + \sum_{s\in[s^*]} n_{s-1}t\right) = \mathcal{O}\left(n_0t + 2n_0t \sum_{s\in[s^*]} \tfrac{1}{2^s}\right) = \mathcal{O}(n_0t),$$

where $\|(T_0, \mathcal{X}_0)\| \leq \mathcal{O}(n_0t)$ due to Proposition 2.32a) was used. □

5.3.5 Improving the Running Time

Consider a nonredundant tree decomposition $(T, \mathcal{X})$ for which $\|(T, \mathcal{X})\|$ is asymptotically smaller than nt, where n denotes the number of vertices of the underlying graph and $t-1$ denotes the width of $(T, \mathcal{X})$. Then, the algorithm in Theorem 5.37 does not run in linear time. So, first one can think about tightening the analysis of this algorithm: The while loop in Algorithm 5.4 is executed at most $\log_2\left(\frac{1}{r(T,\mathcal{X})}\right) + 1$ times due to (5.19) in the proof of Theorem 5.21. As the size of the current tree decomposition does not increase, this yields a running time of $\mathcal{O}\left(\|(T, \mathcal{X})\| \cdot \log_2\left(\frac{1}{r(T,\mathcal{X})}\right)\right)$. Since we have no control over the amount by which the size of the tree decomposition decreases in each round of the while loop, the analysis cannot be improved further. In order to achieve a linear running time, one execution of the while loop needs to be done in less than $\mathcal{O}(\|(T, \mathcal{X})\|)$ time. So the algorithm cannot traverse the entire tree decomposition to compute a restriction and a new heaviest path in each execution of the while loop, as these computations both need to traverse the entire tree decomposition. This section presents an implementation of an algorithm computing an exact cut with the properties of Theorem 5.21 in linear time.

Theorem 5.39 (improved algorithmic version of Theorem 5.21).
For every graph G on n vertices with $V(G) = [n]$, every integer $m \in [n]$, and every tree decomposition $(T, \mathcal{X})$ of G of width at most $t-1$, an m-cut (B, W) in G with

$$e_G(B, W) \leq \frac{t\Delta(G)}{2}\left(\left(\log_2\left(\frac{1}{r(T,\mathcal{X})}\right)\right)^2 + 9\log_2\left(\frac{1}{r(T,\mathcal{X})}\right) + 8\right)$$

can be computed in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time and requires only the tree decomposition $(T, \mathcal{X})$ as input.

As explained above, the reason that the previous implementation does not run in linear time is that the algorithm traverses the tree decomposition too often. So consider one execution of the while loop in Algorithm 5.4. Let $(T, \mathcal{X})$ be the current tree decomposition and denote by G the underlying graph. Furthermore, let $P \subseteq T$ be the path considered in the beginning of the while loop. Assume that Option 2) in Lemma 5.38 occurs. Then, $(T, \mathcal{X})$ is used in Line 6 and to compute a tree decomposition $(T', \mathcal{X}')$ of $G[Z]$ and a heaviest path in T' in Line 8. To execute Line 6, Algorithm 5.6 is applied and there, the tree decomposition $(T, \mathcal{X})$ is used for the computation of the set of P-parameters in Line 1 and the approximate cut in Line 23. The next lemma is an advanced version of Lemma 5.38, that avoids to use the tree decomposition as much as possible. The main idea is to not compute a heaviest path in $(T', \mathcal{X}')$, but to use a piece of P. Then, the main algorithm can adjust the set of P-parameters of G to be the set of P'-parameters of G' with respect to some suitable tree decomposition $(T', \mathcal{X}')$ of G', which is kept implicit. Therefore, the algorithm only needs a tree decomposition when applying Lemma 4.8 in Line 23. Recall that there it does not need a tree decomposition of the graph G itself but only of a subgraph $G[S_i]$ for some $i \in V(P)$. The next lemma shows that a suitable tree decomposition of $G[S_i]$ can be computed from the input tree decomposition $(T_0, \mathcal{X}_0)$, which is a tree decomposition of some supergraph of G. Therefore, the new version of Lemma 5.22 uses three graphs: G and $G' = G[Z]$, which are known from the previous version, and the graph G_0, which does not appear in the previous version. The tree decompositions of G and G' are not computed explicitly. Whenever the lemma refers to a tree decomposition of G, as for example the tree decomposition to which the set of P-parameters of G refers, then it refers to the restriction of $(T_0, \mathcal{X}_0)$ to some suitable tree T and the graph G. All in all, we will show that Algorithm 5.4 can be implemented such that in each execution of the while loop, it only needs a small part of the implicit tree decomposition $(T, \mathcal{X})$, whose decomposition tree is disjoint from T', where $(T', \mathcal{X}')$ is the implicit tree decomposition of G'. Hence, in all executions of the while loop, Algorithm 5.4 traverses the input tree decomposition $(T_0, \mathcal{X}_0)$ only once.

Lemma 5.40 (improved algorithmic version of Lemma 5.22).
Consider an arbitrary graph G_0 on n_0 vertices with $V(G_0) = [n_0]$ and a nonredundant tree decomposition $(T_0, \mathcal{X}_0)$ of G_0 of width at most $t-1$ and with $\mathcal{X}_0 = (X_0^i)_{i \in V(T_0)}$. Let G be a subgraph of G_0 on n vertices, $m \in [n]$ an arbitrary integer, T a tree with $V(T) \subseteq V(T_0)$ such that the restriction $(T, \mathcal{X})$ of $(T_0, \mathcal{X}_0)$ to T and G is a tree decomposition of G, and $P \subseteq T$ a nonredundant path with respect to $\mathcal{X}$. Then, there is an algorithm that computes the lists of the vertices in the sets B and Z of a cut (B, W, Z) in G that satisfies one of the following properties:

1) $|B| = m$, $Z = \emptyset$, $e_G(B, W) \leq 2t\Delta(G)$, and the algorithm takes $\mathcal{O}(n)$ time, or

2) $|B| \leq m \leq |B| + |Z|$ with $0 < |Z| \leq \frac{1}{2}n$, $e_G(B, W, Z) \leq t\Delta(G)\log_2\left(\frac{16}{w^*_{\mathcal{X}}(P)}\right)$, and there is a tree T' with $V(T') \subseteq V(T)$ such that the restriction $(T', \mathcal{X}')$ of $(T_0, \mathcal{X}_0)$ to T' and $G[Z]$ is a tree decomposition of $G[Z]$, and moreover, T' contains a nonredundant path P' with respect to $\mathcal{X}'$ that satisfies $w^*_{\mathcal{X}'}(P') \geq 2w^*_{\mathcal{X}}(P)$. In this case, the algorithm takes $\mathcal{O}\left(n + \sum_{i \in V(T) \setminus V(T')} |X_0^i|\right)$ time and also modifies the set of P-parameters to be the set of P'-parameters for $G[Z]$ with respect to $(T', \mathcal{X}')$.

The algorithm requires the tree decomposition $(T_0, \mathcal{X}_0)$, the size-parameter m, and the set of P-parameters for the graph G with respect to $(T, \mathcal{X})$ as input. In both cases, the tree decomposition $(T_0, \mathcal{X}_0)$ is not modified during the computations.

Note that the algorithm in the previous lemma uses only the set of P-parameters for the graph G and does not require the graph G itself as input. Therefore, it does not need to compute the graph $G[Z]$ but only the set of P'-parameters for $G[Z]$ in order to apply the lemma iteratively to $G[Z]$ if Option 2) occurs, as done ahead. In this case, the set of P'-parameters contains all necessary information about $G[Z]$ as the array A_V provides a list of the vertices in the set Z. Before going into the technical details of the proof of Lemma 5.40, it is shown how to employ it in order to derive the linear-time algorithm in Theorem 5.39.

Proof of Theorem 5.39. Consider a graph G and a tree decomposition $(T, \mathcal{X})$ of G. Denote by n the number of vertices of G, assume that $V(G) = [n]$, and fix an integer $m \in [n]$. We will apply Algorithm 5.4 and use the algorithm contained in Lemma 5.40 in Line 6. First, this implementation of Algorithm 5.4 returns an m-cut with the desired properties, as the cut (B, W, Z), the graph $G[Z]$, the tree decomposition $(T', \mathcal{X}')$, and the path P' in Lemma 5.40 have the same properties as in Lemma 5.22, which was used when proving Theorem 5.21.

The implementation described here, only requires the tree decomposition $(T, \mathcal{X})$ and the size-parameter m as input, but not the graph G. Therefore, in the following, Line 2 is ignored and, in Line 7, the update of the graph G is ignored. Moreover, the copy G_0 of G was only used to state the returned cut and is not needed in the implementation as only an unordered list of the vertices in the set B is computed. In Line 3, the algorithm additionally computes the set of P-parameters of G with respect to $(T, \mathcal{X})$. Lines 1-4 take $\mathcal{O}(\|(T, \mathcal{X})\|)$ time by Proposition 5.36, Lemma 5.32, and Lemma 5.35. Denote by $(T_0, \mathcal{X}_0)$ the tree decomposition $(T, \mathcal{X})$ after Line 4 has been executed and let $\mathcal{X}_0 = (X_0^i)_{i \in V(T_0)}$. Note that $\|(T_0, \mathcal{X}_0)\| \leq \|(T, \mathcal{X})\|$ and $r(T_0, \mathcal{X}_0) \geq r(T, \mathcal{X})$ by Proposition 5.36. Furthermore, let s^* be the number of executions of the while loop and, for $s \in [s^* - 1]$, denote by G_s the implicit graph G and by $(T_s, \mathcal{X}_s)$ the implicit tree decomposition $(T, \mathcal{X})$ of G after the s^{th} execution of the while loop. Let n_s be the number of vertices of G_s for $s \in [s^* - 1] \cup \{0\}$, where G_0 is the underlying graph of the input tree decomposition $(T, \mathcal{X})$. For technical reasons set $V(T_{s^*}) = \emptyset$. Lemma 5.40 implies that, for every $s \in [s^*]$, Lines 6-8 take $\mathcal{O}\left(n_{s-1} + \sum_{i \in V(T_{s-1}) \setminus V(T_s)} |X_0^i|\right)$ time in the s^{th} execution of the while loop. Indeed, if $s \neq s^*$, the algorithm contained in Lemma 5.40 returns the set of P'-parameters for the graph $G' = G[Z]$, which are needed for the next execution of the while loop, and the algorithm does not modify the tree decomposition $(T_0, \mathcal{X}_0)$ when Option 2) occurs. Furthermore, the algorithm keeps track of the size of the set B such that the parameter $\tilde{m}$ in Line 6 can be computed in constant time. The union in Line 7 takes $\mathcal{O}(1)$ time because it is a disjoint union by invariant (ii) in the proof of Theorem 5.21 and the set B is stored as an unordered list. Recall that (vi) from the proof of Theorem 5.21 implies that $n_s \leq \frac{1}{2^s} n$ for every $s \in [s^* - 1] \cup \{0\}$. It follows that

$$\sum_{s=1}^{s^*} \sum_{i \in V(T_{s-1}) \setminus V(T_s)} \left|X_0^i\right| \leq \sum_{i \in V(T_0)} \left|X_0^i\right|,$$

as $V(T_s) \subseteq V(T_{s-1})$ for all $s \in [s^*]$. Therefore, the total running time is bounded by

$$\begin{aligned}
\mathcal{O}\left(\|(T, \mathcal{X})\|\right) \;+\; & \sum_{s=1}^{s^*} \mathcal{O}\left(n_{s-1} + \sum_{i \in V(T_{s-1}) \setminus V(T_s)} \left|X_0^i\right|\right) \\
= \; & \mathcal{O}\left(\|(T, \mathcal{X})\| + \left(\sum_{s=1}^{s^*} \frac{n}{2^{s-1}}\right) + \left(\sum_{i \in V(T_0)} \left|X_0^i\right|\right)\right) \\
= \; & \mathcal{O}\left(\|(T, \mathcal{X})\| + n + \|(T_0, \mathcal{X}_0)\|\right) \;=\; \mathcal{O}(\|(T, \mathcal{X})\|). \qquad \square
\end{aligned}$$

It remains to prove Lemma 5.40.

Proof of Lemma 5.40. Fix an arbitrary graph G_0 on n_0 vertices with $V(G_0) = [n_0]$ and a nonredundant tree decomposition $(T_0, \mathcal{X}_0)$ of G_0 of width at most $t - 1$ and with $\mathcal{X}_0 = (X_0^i)_{i \in V(T_0)}$. Furthermore, fix an arbitrary subgraph $G \subseteq G_0$ on n vertices, an integer $m \in [n]$, a tree T with $V(T) \subseteq V(T_0)$ such that the restriction $(T, \mathcal{X})$ of $(T_0, \mathcal{X}_0)$ to T and G is a tree decomposition of G, and an arbitrary nonredundant path $P \subseteq T$ with respect to $\mathcal{X}$. Recall that the graphs G_0 and G are implicit. The input of the algorithm contained in Lemma 5.40 is the tree decomposition $(T_0, \mathcal{X}_0)$ of the implicit graph G_0, the size-parameter m, and the set of P-parameters for the graph G.

The aim is to argue that, among others, a partition (B, W, Z) of $V(G)$ with the properties stated in Lemma 5.40 can be computed in the time stated there. More precisely, when Option 2) occurs, then a suitable path $P' \subseteq T'$ and the set of P'-parameters for the graph $G[Z]$ are computed, with respect to an implicit tree decomposition $(T', \mathcal{X}')$ of $G[Z]$. In the proof of Lemma 5.22, an earlier version of Lemma 5.40 that does not contain an algorithm, the restriction of $(T, \mathcal{X})$ to T itself and $G[Z]$ was used to obtain a suitable tree decomposition $(T', \mathcal{X}')$ of $G[Z]$. As argued above, when Case 2) from the proof of Lemma 5.22 occurs, i. e., the algorithm constructs a partition with the properties in Option 2) of Lemma 5.40, it cannot work with the restriction of $(T, \mathcal{X})$ and a heaviest path in T' with respect to $\mathcal{X}'$ anymore when the algorithm is supposed to run in the time stated in Lemma 5.40. Therefore, we first propose a new construction for a suitable tree T' and a suitable path P', and show that T' and P' satisfy the properties in Option 2) in Lemma 5.40. Other than that, the construction and notation used in the proof of Lemma 5.22 is kept. See also Algorithm 5.6 in the proof of Lemma 5.38, the first algorithmic version of Lemma 5.22. In particular, the partition (B, W, Z) is still defined in the same way. Let $P = (V_P, E_P)$ and $T = (V_T, E_T)$. Moreover, let $\mathcal{X} = (X^i)_{i \in V(T)}$ and note that $X^i = X_0^i \cap V(G)$ for all $i \in V(T)$. Recall the definitions of the ends i_0 and j_0 of P, the graph T^+ as well as, for each $i \in V_P$, the tree T_i, the sets R_i, S_i, U_i^b, H_i^b, P_i^b, U_i^f, H_i^f, P_i^f, and the nodes i^b, i_ℓ^b, i^f, i_ℓ^f, see also Table 5.2 on Page 178. Note that all definitions are with respect to the tree decomposition $(T, \mathcal{X})$, which is implicit in the implementation described here.

Fix a doubling node $i \in V_P$ and recall that, in the proof of Lemma 5.38, it was argued that i satisfies one of the properties required for Case 2a) or Case 2b) in the proof of Lemma 5.22. Assume that i satisfies the property for Case 2a). Note that $P_i^b \neq \emptyset$ by Proposition 5.29a) as i is b-special, and that P_i^b induces a path with ends i^b and i_ℓ^b, or a cycle in T^+. Next, a tree decomposition $(T', \mathcal{X}')$ of the graph $G[Z]$ with $Z := U_i^b \cup H_i^b$ is constructed. Let H_T be the subgraph of T that is induced by P_i^b and the nodes of T_h for all $h \in P_i^b \setminus \{i^b\}$. If P_i^b induces a connected subgraph in T, then H_T is connected and we define $T' = H_T$ and $i_0' := i^b$, see also Figure 5.17a). Otherwise, P_i^b induces two paths in T, one with ends i^b and j_0 and the other with ends i_0 and i_ℓ^b. In this case, H_T consists of two components and we define T' to be the tree obtained from H_T by adding the edge $\{i_\ell^b, i^b\}$ and we set $i_0' := i_0$, see Figure 5.17b). Furthermore, let P' be the path induced by P_i^b in T' and note that i_0' is an end of P'.

If i does not satisfy the property required for Case 2a) in the proof of Lemma 5.22, then i must satisfy the property required for Case 2b). So, assume that i is f-special. Then, P_i^f is not empty by Proposition 5.29a) and P_i^f either induces a path or a cycle in T^+. Let H_T be the subgraph of T that is induced by P_i^f and the nodes of T_h with $h \in P_i^f \setminus \{i^f\}$. Analogously to the case when i is b-special, the tree T' is defined in the following way: If P_i^f induces a connected subgraph in T, then H_T is connected and we define $T' := H_T$ and $i_0' := i^f$. Otherwise, P_i^f induces two paths in T, one with ends i^f and j_0 and one with ends i_0 and i_ℓ^f. Then, T' is defined to be the tree obtained from H_T by adding the edge $\{i_\ell^f, i^f\}$, and $i_0' := i_0$. Furthermore, let P' be the path induced by P_i^f in T' and note that i_0' is an end of P'.

Let $(T', \mathcal{X}')$ be the restriction of $(T, \mathcal{X})$ to T' and $G[Z]$, i.e., the cluster associated with node $j \in V(T')$ is $X^j \cap Z$. Note that, since $V(T') \subseteq V(T) \subseteq V(T_0)$ and since $(T, \mathcal{X})$ is the restriction of $(T_0, \mathcal{X}_0)$ to T and G, the tuple $(T', \mathcal{X}')$ is also the restriction of $(T_0, \mathcal{X}_0)$ to T' and $G[Z]$, as desired for Option 2). The next claim shows that the tuple $(T', \mathcal{X}')$ is indeed a tree decomposition and that the tree decomposition $(T', \mathcal{X}')$ and the path P' have the desired properties for Option 2).

Claim 5.41.

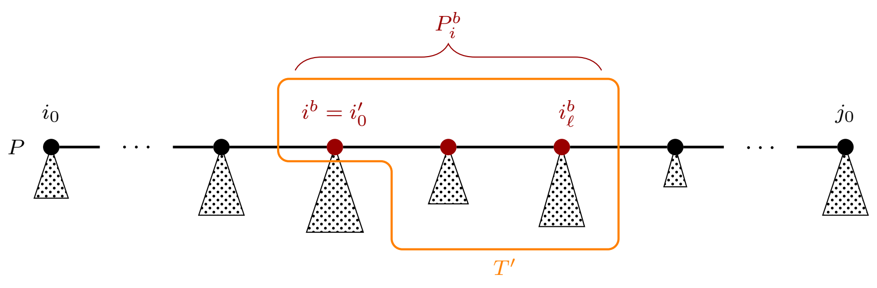
a) $(T', \mathcal{X}')$ is a tree decomposition of $G[Z]$.

b) The end i_0' is a nonredundant end of P' with respect to $\mathcal{X}'$ and $w^*_{\mathcal{X}'}(P') \geq 2w^*_{\mathcal{X}}(P)$.

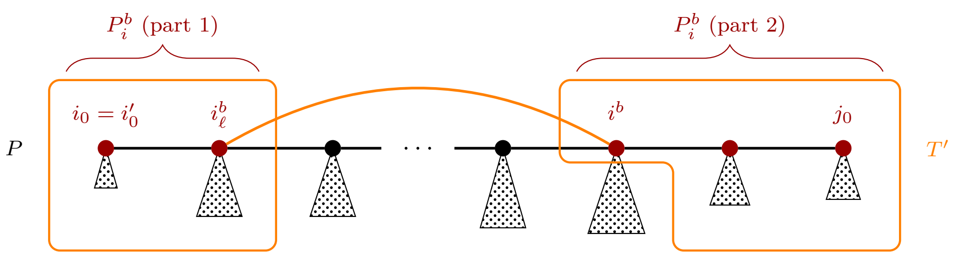
c) T' contains no node in $V(T_i) \setminus \{i\}$.

Indeed, assume that the node i that was picked when defining T' is b-special. The case when i is f-special is analogous. To show Part a), observe first that each cluster in $\mathcal{X}'$ is a subset of Z and it is easy to see that T' is a tree. It is now argued that $(T', \mathcal{X}')$ satisfies the properties (T1), (T2), and (T3). For (T1), recall that Proposition 5.29a) implies that $U_i^b = \bigcup_{j \in P_i^b} R_j$. Therefore,

$$\begin{aligned} Z &= U_i^b \cup H_i^b = R_{i^b} \cup \left(\bigcup_{j \in P_i^b \setminus \{i^b\}} (R_j \cup S_j) \right) \\ &\subseteq X^{i^b} \cup \left(\bigcup_{j \in P_i^b \setminus \{i^b\}} \bigcup_{h \in V(T_j)} X^h \right) = \bigcup_{j \in V(T')} X^j \end{aligned}$$



a) When H_T is connected.



b) When H_T is disconnected.

Figure 5.17: Proof of Lemma 5.40. Construction of the tree T' for Option 2) when the doubling node i has the property required for Case 2a) in the proof of Lemma 5.22.

and every vertex in Z is in some cluster in $\mathcal{X}'$. To show that $(T', \mathcal{X}')$ satisfies (T2), let $\{x, y\}$ be an arbitrary edge in $G[Z]$.

Case i) $\{x, y\} \not\subseteq R$. Without loss of generality assume that $x \notin R$. Denote by h_x the path-node of x and note that $h_x \neq i^b$. Then,

$$I_x := \{h \in V(T) \colon x \in X^h\} \subseteq V(T_{h_x}) \setminus \{h_x\} \subseteq V(T'),$$

as $(T, \mathcal{X})$ satisfies (T3') and $x \notin X^{h_x}$. Since (T2) is satisfied for $(T, \mathcal{X})$, each neighbor of x in G, and in particular the vertex y, must be in some cluster X^h with $h \in I_x \subseteq V(T')$ and therefore $(T', \mathcal{X}')$ satisfies (T2) for the edge $\{x, y\}$.

Case ii) $\{x, y\} \subseteq R$. For two distinct nodes $h \in V_P$ and $h' \in V_P$ we say that h appears before h' on P, if i_0 is closer to h than to h'. Denote the path-nodes of x and y by h_x and h_y, respectively. Note that $x \in X^{h_x}$ and $y \in X^{h_y}$ as well as that $h_x \in P_i^b \subseteq V(T')$ and $h_y \in P_i^b \subseteq V(T')$ as $x, y \in Z \cap R = U_i^b$. If $h_x = h_y$, then there is nothing more to prove, since $\{x, y\} \subseteq X^{h_x} \cap Z$. So assume that $h_x \neq h_y$ and without loss of generality assume that h_x appears before h_y on P. For every $h \in V(P) \setminus \{h_y\}$ that appears before h_y on P, the vertex y does not lie in X^h by the definition of the path-node of y and additionally, for every $h' \in V(T_h)$, the vertex y does not lie in $X^{h'}$ as $(T, \mathcal{X})$ satisfies (T3'). Therefore, $x \in X^{h_y}$ as $(T, \mathcal{X})$ satisfies (T2) for the edge $\{x, y\}$ and (T3') for x. This implies that $x \in X^{h_y} \cap Z$ and $y \in X^{h_y} \cap Z$ and, hence, $(T', \mathcal{X}')$ satisfies (T2) for the edge $\{x, y\}$.

Now, consider the property (T3). If every edge of T' is an edge of T, i. e., T' is a subgraph of T, then every path in T' is also a path in T and $(T', \mathcal{X}')$ satisfies (T3) because $(T, \mathcal{X})$ satisfies (T3). Otherwise, T' contains exactly one edge that is not in T, namely the edge $e = \{i^b_\ell, i^b\}$. In that case, as the unique i^b_ℓ,i^b-path in T uses no edge in T', for all i', j', h' in T', the node h' is on the unique i',j'-path in T'

only if h' is on the unique i',j'-path in T, see Figure 5.17b). Hence, $(T', \mathcal{X}')$ satisfies (T3), as $(T, \mathcal{X})$ satisfies (T3).

To show Part b) of Claim 5.41, consider first an arbitrary vertex $x \in U_i^b \subseteq Z$. Let h be the path-node of x with respect to the path P, its nonredundant end i_0, and the tree decomposition $(T, \mathcal{X})$. Then, h lies in $P_i^b = V(P')$ due to Proposition 5.29a) and h is also the path-node of x when defining the path-node of x with respect to the path P', its end i_0', and the tree decomposition $(T', \mathcal{X}')$. Hence, when defining sets R_j' for $j \in V(P')$ analogous to the sets R_j for $j \in V_P$ but with respect to the tree decomposition $(T', \mathcal{X}')$, the path P', and its end i_0', then $R_j' = R_j$ for all $j \in V(P')$. Proposition 5.25a) implies that $R_{i_0'}$ and, thus, also the cluster corresponding to i_0' in $\mathcal{X}'$ is non-empty. If P' consists only of the node i_0', it already follows that i_0' is a nonredundant end of P' with respect to $\mathcal{X}'$. Otherwise, let $j \neq i_0'$ be a node in P' and let j' be the node before j on P'. Denote by Y and Y' the clusters of j and j' in $\mathcal{X}'$ respectively. The definition of the path-node for each vertex $x \in R_j'$ implies that $R_j' \subseteq Y \setminus Y'$ and, hence, $Y \setminus Y'$ is not empty as $R_j' = R_j$ is not empty by Proposition 5.25a). Consequently, the end i_0' of P' and the path P' itself are nonredundant with respect to $\mathcal{X}'$. As all sets R_j with $j \in V_P$ are pairwise disjoint by Proposition 5.25b) and $R_j' = R_j$ is contained in the cluster of j in $\mathcal{X}'$ for all $j \in V(P')$, the path P' satisfies $w_{\mathcal{X}'}(P') \geq |U_i^b|$. Recalling that (5.29) in the proof of Lemma 5.22 implied $|Z| = |U_i^b| + |H_i^b| \leq \frac{1}{2r}|U_i^b|$, where $r := w_{\mathcal{X}}^*(P)$, yields $w_{\mathcal{X}'}^*(P') \geq \frac{|U_i^b|}{|Z|} \geq 2w_{\mathcal{X}}^*(P)$.

To show Part c) of Claim 5.41, for a contradiction, assume that there is a node $j \in V(T') \cap (V(T_i) \setminus \{i\})$. Recall that $V(T') = \{i^b\} \cup \{V(T_h)\colon\ h \in P_i^b \setminus \{i^b\}\}$. Then, i would lie in P_i^b and $i \neq i^b$, which contradicts Proposition 5.29d). This completes the proof of Claim 5.41.

After presenting the new construction for $(T', \mathcal{X}')$ and P', it is explained now how to implement the algorithm contained in Lemma 5.40. Recall that the algorithm receives as input the tree decomposition $(T_0, \mathcal{X}_0)$ of the graph G_0, the size-parameter m, and the set of P-parameters for the implicit graph G, which is with respect to the implicit tree decomposition $(T, \mathcal{X})$. The arrays and the list of the set of P-parameters of G are denoted by A_L, A_V, A_R, A_P, and L_P, respectively. Recall that we assumed that $V(G_0) = [n_0]$ and, hence, A_L, A_R, and A_P are each of length n_0. Basically, the algorithm in Lemma 5.40 follows the construction from the proof of Lemma 5.22, except for the tree decomposition $(T', \mathcal{X}')$ and the path P' if Case 2) occurs. This construction was also described in Algorithm 5.6 in the proof of Lemma 5.38. In the following, the line numbers refer to Algorithm 5.6. However, now the situation is slightly different: The algorithm does not receive the tree decomposition $(T, \mathcal{X})$ as input and the vertices of G are not renamed, which means that $V(G) = [n]$ cannot be assumed. The algorithm now receives the set of P-parameters of G and, hence, it skips Line 1, where the set of P-parameters for the graph G is computed. Furthermore, it does not return a list of the vertices in the set W and, therefore, also skips Line 26. Lines 2-25 rely only on the set of P-parameters of G, except for the computation of a c-approximate cut in Line 23. Only a few modifications are needed to achieve the desired running time for Lines 2-22 and Lines 24-25: First, the algorithm cannot traverse the entire array A_R, which used to be of length n in the first implementation and now is of length n_0. Thus, whenever the first implementation traversed the array A_R, the new algorithm traverses all vertices v stored in the array A_V and only accesses the entries $A_R[v]$. As A_V is of length n and every vertex of G appears somewhere in A_V, this provides a method to traverse all vertices in R in $\mathcal{O}(n)$ time. If Case 1) of the proof of Lemma 5.22 occurs, i. e., the algorithm constructs a cut (B, W, Z) with the properties in Option 1) and Lines 3-4 are executed, then there is nothing else to do and the running time is proportional to n, as desired.

From now on, assume that Case 2) of the proof of Lemma 5.22 occurs, i. e., the algorithm needs to construct a cut with the properties in Option 2) and it executes Lines 6-25. In the first implementation, the algorithm created a few arrays of length $n_T := V(T)$, for example to store the numbers $u^b(i)$, $h^b(i)$, and so on when executing Lines 6-8. This is not applicable here, as the algorithm does not have the tree T, which is kept implicit, and also T might have more than n nodes as $(T, \mathcal{X})$ is not necessarily nonredundant. Recall that, in each of these arrays, only the entries corresponding to the nodes in P are used and the algorithm can access the list L_P from the set of P-parameters, which contains all nodes in P in the order in which they occur when traversing P from i_0 to j_0. Therefore, all values computed in Lines 6-9 can be stored in arrays of length $|L_P|$, where the i^{th} entry corresponds to the i^{th} node in L_P. A disadvantage of the shorter arrays is that, for example when accessing the value $u^b(i)$ for some specific node $i \in V_P$, the algorithm first needs to find out the entry that corresponds to i. For executing Lines 6-8, this is okay, as the values computed there are increased in exactly the order in which the nodes appear on P. For example, when the algorithm computes all values $s(i)$ in Line 6, it goes through all vertices in V in order of increasing labels and therefore, the algorithm has to access the entries $s(i)$ in the order in which the nodes i appear in the list L_P. This is similar when computing the values $u^b(i)$ for all $i \in V_P$, except that the algorithm might have to start traversing L_P from the beginning for a second time after reaching j_0. Hence, $\mathcal{O}(|L_P|)$ additional time is needed to execute Lines 6-9 and the term $\mathcal{O}(n_T)$ disappears, as the initialization of the arrays now takes $\mathcal{O}(|L_P|)$ time. Hence, Lines 6-9 now take time proportional to $|L_P| + n$. When executing Line 10, the algorithm also determines the position of this node i in the list L_P and, hence, no further time is lost due to the different storage. Recall (5.31) and note that the computation of the lists of the vertices in the sets B and Z, except for Line 23, takes time proportional to $n + |L_P| \leq 2n$ in the new implementation. Indeed, the sets R_i with $i \in V_P$ form a partition of R where each set is nonempty due to Proposition 5.25. Therefore, $|L_P| \leq |R| \leq |V(G)|$.

Next, the implementation of Line 23 is discussed. To apply the procedure contained in Lemma 4.8 to the graph $G[S_i]$, the algorithm needs to provide the tree decomposition $(\tilde{T}, \tilde{\mathcal{X}})$ of $G[S_i]$, which was defined as the restriction of $(T, \mathcal{X})$ to $\tilde{T} = T_i$ and $G[S_i]$ in the proof of Lemma 5.22. Note that the algorithm does not store the tree decomposition $(T, \mathcal{X})$, but the algorithm can compute $(\tilde{T}, \tilde{\mathcal{X}})$ nevertheless as $(\tilde{T}, \tilde{\mathcal{X}})$ is also the restriction of $(T_0, \mathcal{X}_0)$ to $\tilde{T}$ and $G[S_i]$ and $(T_0, \mathcal{X}_0)$ is stored. To compute $(\tilde{T}, \tilde{\mathcal{X}})$, recall that the set of P-parameters contains the tree T_i. For the clusters, let $\tilde{\mathcal{X}} = (\tilde{X}^h)_{h \in V(\tilde{T})}$. As $S_i \cap X_0^i = S_i \cap X^i = \emptyset$, the cluster of node i in $\tilde{\mathcal{X}}$ is empty. To compute the other clusters, the algorithm applies the procedure in Proposition 2.31, because Proposition 5.34a) and b) together provide a method to determine in constant time whether a vertex $v \in [n_0]$ is in S_i. Computing the clusters in $\tilde{\mathcal{X}}$ takes time proportional to

$$1 \; + \; \left|V(\tilde{T}) \setminus \{i\}\right| \; + \sum_{h \in V(\tilde{T}) \setminus \{i\}} \left|X_0^h\right| \;\; \leq \;\; 1 \; + \; 2 \sum_{h \in V(\tilde{T}) \setminus \{i\}} \left|X_0^h\right|,$$

because $X_0^h \neq \emptyset$ for all $h \in V(T_0)$ as $(T_0, \mathcal{X}_0)$ is nonredundant and because it suffices to search in the tree $\tilde{T}$, i. e., the tree T_0 does not need to be traversed. Observe that the algorithm does not modify the tree decomposition $(T_0, \mathcal{X}_0)$ during this process when storing $(\tilde{T}, \tilde{\mathcal{X}})$ in a new place instead of overwriting $(T_0, \mathcal{X}_0)$. While computing the clusters in $\tilde{\mathcal{X}}$, the algorithm keeps track of the vertex $v_s \in S_i$ with the smallest label among all vertices in S_i to set up a bijection between the vertices in S_i and $[s]$ for $s := |S_i|$, as in the first implementation. This is required when applying the procedure in Lemma 4.8, which takes

$$\mathcal{O}\left(\|(\tilde{T}, \tilde{\mathcal{X}})\|\right) = \mathcal{O}\left(\left|V(\tilde{T})\right| + \sum_{h \in V(\tilde{T}) \setminus \{i\}} \left|\tilde{X}^h\right|\right)$$

time. The running time of the procedure in Lemma 4.8 simplifies to $\mathcal{O}\left(\sum_{h \in V(T) \setminus V(T')} \left|X_0^h\right|\right)$, because $X_0^h \neq \emptyset$ for all $h \in V(T_0)$ as mentioned above and because $V(\tilde{T}) \setminus \{i\} = V(T_i) \setminus \{i\}$ and $V(T')$ are disjoint by Claim 5.41c).

This completes the description of computing the cut (B, W, Z) in the implicit graph G, or more precisely the lists of the vertices in B and Z. Other than that, the algorithm also needs to adjust the set of P-parameters of the graph G to be the set of P'-parameters of $G[Z]$. As the set Z depends on whether the doubling node i chosen in Line 10 is b-special or f-special, the adjustment of the set of P-parameters depends on this as well. Assume for now that i is b-special and recall that in this case

$$Z = H_i^b \cup U_i^b = \left\{ v \in V\colon\ L(v) \text{ is between } L(x) \text{ and } L(y) \right\},$$

where x denotes the vertex in R_{i^b} with the smallest label and y denotes the vertex in $R_{i^b_\ell}$ with the largest label, as in the proof of Lemma 5.22. So, in order to adjust the arrays A_L and A_V for the P'-labeling, the algorithm only needs to shift the labels cyclically. To do so, it first computes the nodes i^b and i^b_ℓ, which are the path-nodes of $N_m^{-1}(x^b(i))$ and $N_m^{-1}(x^b_\ell(i))$, respectively, where $x^b(i)$ and $x^b_\ell(i)$ are the vertices computed in Line 7, and $N_m^{-1}(x^b(i))$ and $N_m^{-1}(x^b_\ell(i))$ coincide with x and y. The algorithm goes through the list L_P and checks whether the node i^b appears before the node i^b_ℓ or $i^b = i^b_\ell$. If one of these happens, the edge $\{i^b_\ell, i^b\}$ was not added when constructing the tree T' and the algorithm deletes all entries in L_P that appear before i^b and all entries that appear after i^b_ℓ in L_P, which yields the list $L_{P'}$ of all vertices on P' in the correct order. Simultaneously, the algorithm deletes the trees T_j for all nodes j that are removed from L_P. Furthermore, the algorithm shifts the labels of the vertices in Z such that $x = N_m^{-1}(x^b(i))$ obtains label 1. Otherwise, i. e., i^b appears after i^b_ℓ in L_P, the edge $\{i^b_\ell, i^b\}$ was added when constructing T' (except for the case when i^b is the node after i^b_ℓ on P, where the tree T already contains the edge $\{i^b_\ell, i^b\}$). Note that, in this case, $i_0 \in P_i^b$ and $j_0 \in P_i^b$ and to obtain the list $L_{P'}$, the algorithm deletes all nodes between i^b_ℓ and i^b, except i^b_ℓ and i^b themselves. Again, the algorithm simultaneously deletes the tree T_j for each node j that is deleted from the list L_P. To adjust the P-labeling, note that the current labels of the vertices in Z are the labels between x and y, which are the labels between x and n and the labels between 1 and y. Hence, to obtain a P'-labeling of the vertices in Z, the algorithm shifts the labels of the vertices with labels between x and n such that x receives label $y + 1$. Note that the trees T_j with $j \in V(P')$ are the same when defined with the tree T and the path P as when they are defined with the tree T' and the path P', except for the tree T_{i^b}, which consists only of the node i^b in the set of P'-parameters and can be adjusted in constant time. The described process of adjusting the collection of trees T_j, the list L_P, and the P-labeling can be done in time proportional to $|L_P| + n$, as the array A_V provides a list of all vertices in G ordered according to their labels. Recall that it was argued above in the proof of Claim 5.41b) that the sets R_i for $i \in V(P')$ are identical when defined with P', its end i'_0, and the tree decomposition $(T', \mathcal{X}')$ and when defined with P, its end i_0, and the tree decomposition $(T, \mathcal{X})$. This is similar for the sets S_i with $i \in V(P')$, except for S_{i^b}, which is empty. Consequently, the arrays A_R and A_P do not need to be adjusted, as the entries referring to vertices not in $G' = G[Z]$ may contain anything. If the doubling node i determined in Line 10 is not b-special then i is f-special and the adjustments of the set of P-parameters can be done analogously to the b-special case in $\mathcal{O}(|L_P| + n)$ time. All together, the adjustment of the set of P-parameters takes $\mathcal{O}(|L_P| + n) = \mathcal{O}(n)$ time, as it was argued above that $|L_P| \le n$.

Summing up, the implementation of the adjusted version of Algorithm 5.6 takes $\mathcal{O}(n)$ time, except the computation of the approximate cut in Line 23, which takes $\mathcal{O}\left(\sum_{h \in V(T) \setminus V(T')} |X_0^h|\right)$ time. Additionally, $\mathcal{O}(n)$ time is needed to adjust the set of P-parameters. Therefore, the desired running time for Option 2) is achieved. □

Chapter 6

Minimum k-Section in Trees and Tree-Like Graphs

This chapter extends some results for bisections to k-sections. Instead of partitioning the vertex set into two sets of equal size, we now aim to partition the vertex set into k sets of equal size, or more precisely, of sizes differing by at most one whenever k does not divide the number of vertices of the graph. On the one hand, we presented results for bisections constructed with separators in Section 3.1 and on the other hand, we constructed bisections in trees with long paths in Chapter 5. For the separator approach, we used separators arising from clusters of a given tree decomposition, and we also used separators from the Planar Separator Theorem when working with planar graphs. Consider a graph G and a bisection (B, W) constructed with separators. Then, the graphs $G[B]$ and $G[W]$ will each allow small bisections, as the same class of separators can be used again. The reason behind this is that every subgraph of a planar graph is planar and that when G allows a tree decomposition of width $t-1$, then we can easily construct a tree decomposition for any subgraph of G of width at most $t-1$ due to Proposition 2.14 and Proposition 2.31. Consequently, when k is a power of 2, it is straightforward to construct k-sections with separators by recursively constructing bisections. For example, Simon and Teng show that every bounded-degree planar graph on n vertices allows a k-section of width $\mathcal{O}(\sqrt{kn})$, see Lemma 4.2 in [ST97]. Moreover, combining their method, see Lemma 4.1 in [ST97], and Theorem 1.11 one can show that, for every $k \geq 2$ that is a power of 2, every graph G on n vertices allows a k-section of width at most

$$\begin{aligned}\sum_{h=0}^{\log_2(k)-1} 2^h(\mathrm{tw}(G)+1)\Delta(G)\log_2\left(\tfrac{n}{2^h}\right) &= (\mathrm{tw}(G)+1)\Delta(G)\big((k-1)\log_2(n) - k\log_2(k) + 2k - 2\big) \\ &\leq (\mathrm{tw}(G)+1)\Delta(G)(k-1)\log_2(n).\end{aligned}$$

Therefore, this chapter first focuses on generalizing the methods used in Section 5.2, where bisections in trees with long paths were constructed. Here, the situation is not as simple as with the separators. Indeed, Section 6.1 discusses the approach of recursively constructing a k-section in a bounded-degree tree and also in bounded-degree trees with linear diameter. It will become clear that Theorem 1.15 about k-sections in trees with linear diameter cannot be obtained by simply reapplying Theorem 1.1 about bisections in trees with linear diameter. Section 6.2 presents the proof for Theorem 1.15 and Section 6.3 generalizes this result to arbitrary graphs with a given tree decomposition.

6.1 Recursive Bisections and Cuts

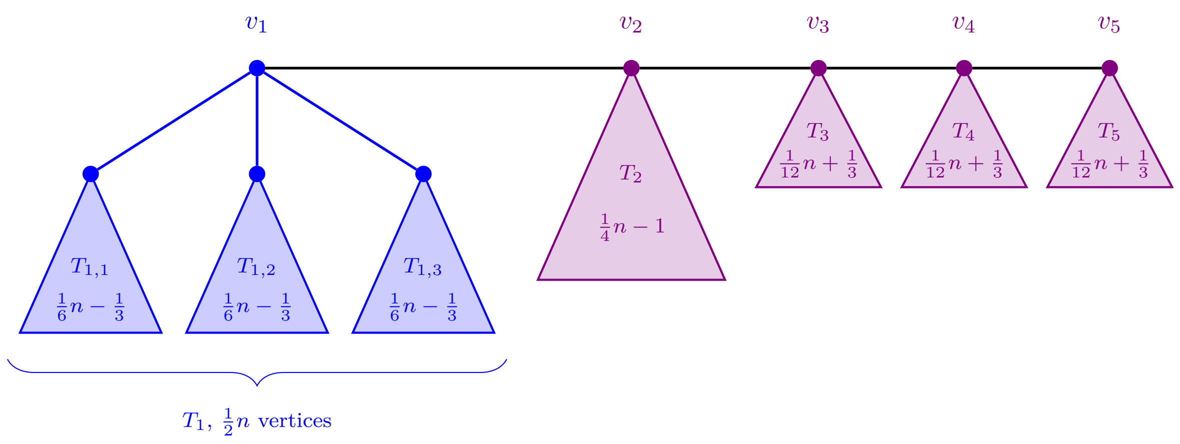
In this section, the approach of constructing a k-section recursively is discussed. This means the following: Assume that $k \geq 2$ is a power of 2 and the aim is to construct a k-section in a graph G of small width, if possible, minimum width. As a tool we want to use a method that computes a minimum bisection. Then, one can first compute a bisection (B, W) in G. Next, one considers the graphs $G[B]$ and $G[W]$ and computes a bisection in each of these graphs, which gives a 4-section (B_1, B_2, B_3, B_4) in G that cuts exactly as many edges as the three bisections that were used. For $k > 4$, one then computes a bisection in $G[B_i]$ for $i \in [4]$ resulting in an 8-section in G and so on, until the desired number of sets is reached. As mentioned in the introduction, this recursive approach can produce a k-section, that cuts much more edges than a minimum k-section, even when using a minimum bisection in each round. In [ST97], Simon and Teng present two examples showing this behavior, one dense and one sparse example. They construct a dense graph G_d and a sparse graph G_s, each on n vertices, that both allow a 4-section of width 12. Both graphs G_d and G_s also allow a bisection of width four, which cannot be obtained by joining the two sets of the unique 4-section of width twelve. In both graphs, the black and the white set of the bisection of width four do not allow bisections of small width. Hence, the recursive approach produces 4-sections of large width. More precisely, it produces a 4-section of width $\Omega(n^2)$ in G_d and a 4-section of width $\Omega(n)$ in G_s, i. e., a constant fraction of all edges of G_d and G_s are cut.

Next, we discuss an example that shows that the approach of constructing a 4-section recursively in a bounded-degree tree can also give a 4-section that cuts asymptotically more edges than a minimum 4-section. Recall that any tree T on n vertices allows a bisection of width $\Delta(T)\log_2(n)$, see also Section 1.1.1 and note that this also follows from Corollary 4.9. So the worst that can happen for bounded-degree trees is that the recursively found 4-section cuts $\Omega(\log_2 n)$ edges and the tree allows a 4-section of constant width. Such an example is presented now. Let $n \geq 4$ be an arbitrary integer such that there is a perfect ternary tree T_1 on $n_1 := \frac{1}{2}n$ vertices and $\frac{1}{2}n$ is even. Note that this is possible as the number $\tilde{n}_h$ of vertices in a ternary tree of height h satisfies $\tilde{n}_{h+1} = 3\tilde{n}_h + 1$ and therefore is alternatingly odd and even. Let v_1 be the root of T_1 and denote by $T_{1,1}$, $T_{1,2}$, and $T_{1,3}$ the isomorphic subtrees of T_1 that are rooted in the three children of the root of T_1. For $i \in [3]$ define $n_{1,i} := \frac{1}{3}\left(\frac{1}{2}n - 1\right) = \frac{1}{6}n - \frac{1}{3}$, which is the number of vertices in $T_{1,i}$. Let T_2 be an arbitrary binary tree on $n_2 := \frac{1}{4}n - 1$ vertices with root v_2 and note that $\frac{1}{4}n$ is an integer. Furthermore, for $i \in \{3, 4, 5\}$, let T_i be a binary tree on $n_i := \frac{1}{4}n - n_{1,i-2} = \frac{1}{12}n + \frac{1}{3}$ vertices and denote by v_i the root of T_i. Joining the roots v_i and v_{i+1} with an edge for all $i \in [4]$ gives a tree on

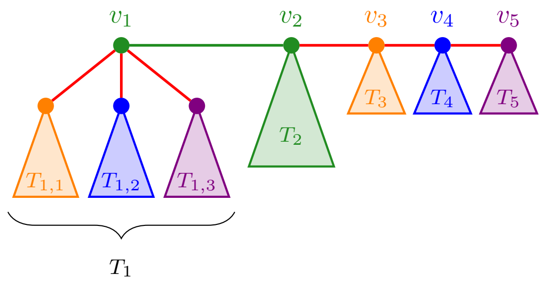
$$\sum_{i=1}^{5} n_i \;=\; \frac{1}{2}n + \frac{1}{4}n - 1 + 3 \cdot \left(\frac{1}{12}n + \frac{1}{3}\right) \;=\; n$$

vertices, see Figure 6.1a). It is easy to see that the maximum degree of T is 4. Furthermore, as $n_{1,i} + n_{i+2} = \frac{1}{4}n$ for $i \in [3]$, the cut (B_1, B_2, B_3, B_4) with $B_i := V(T_{1,i}) \cup V(T_{i+2})$ for $i \in [3]$ and $B_4 := V(T_2) \cup \{v_1\}$ is a 4-section in T of width 6, see Figure 6.1b). Consequently, $\text{MinSec}_4(T) \leq 6$. So let us now recursively construct a 4-section in T. First, T allows exactly one minimum bisection, which is (B, W) with $B := V(T_1)$ and $W := V(T) \setminus B$ and satisfies $e_G(B, W) = 1$, see Figure 6.1c). Now, in the next round of the recursive approach a bisection in $T[B] = T_1$ has to be found, which will cut at least $\Omega(\log_2 n)$ edges by Theorem 2.4.

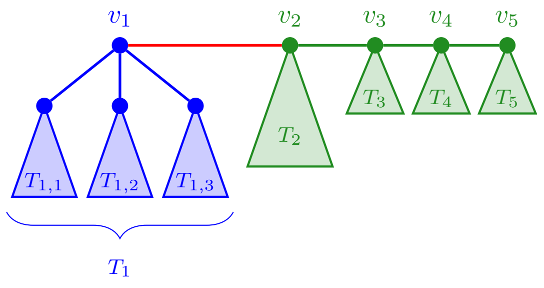
One of our aims in this chapter is to find a k-section of small width in a bounded-degree tree with linear diameter. As the usual choice of the binary trees $T_2, \ldots, T_5$ in the previous example yields a tree T with diameter $\mathcal{O}(\log n)$, one could hope that the situation improves when the diameter of the considered tree is linear. However, one can also choose $T_2, \ldots, T_5$ to be paths, which results in a tree T with $\text{diam}(T) > \frac{1}{4}n$. Since there were no further assumptions on the structure of $T_2, \ldots, T_5$, the above analysis still applies.


v_1
v_2
v_3
v_4
v_5
$T_{1,1}$ $\frac{1}{6}n - \frac{1}{3}$
$T_{1,2}$ $\frac{1}{6}n - \frac{1}{3}$
$T_{1,3}$ $\frac{1}{6}n - \frac{1}{3}$
T_2 $\frac{1}{4}n - 1$
T_3 $\frac{1}{12}n + \frac{1}{3}$
T_4 $\frac{1}{12}n + \frac{1}{3}$
T_5 $\frac{1}{12}n + \frac{1}{3}$
T_1, $\frac{1}{2}n$ vertices

a) Subtrees colored in blue are ternary trees, subtrees colored in purple are binary trees. The number in each subtree indicates its number of vertices.



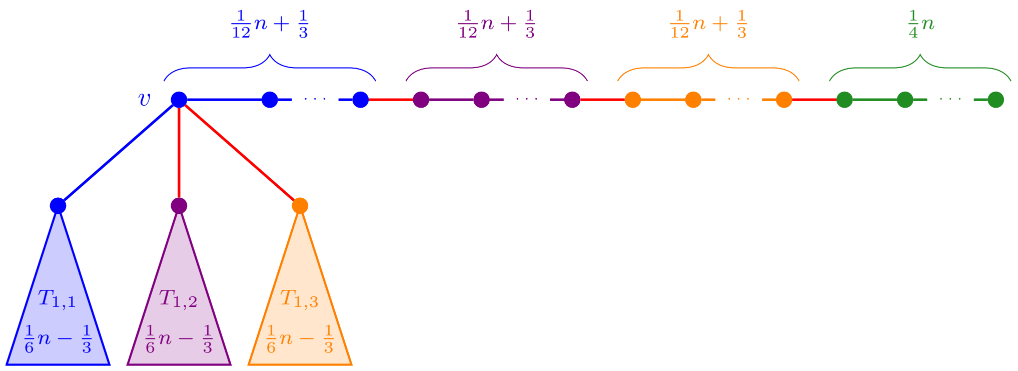
b) Subtrees are colored in orange, blue, violet, and green such that each color indicates one set of a 4-section of width 6. Edges colored in red are cut.



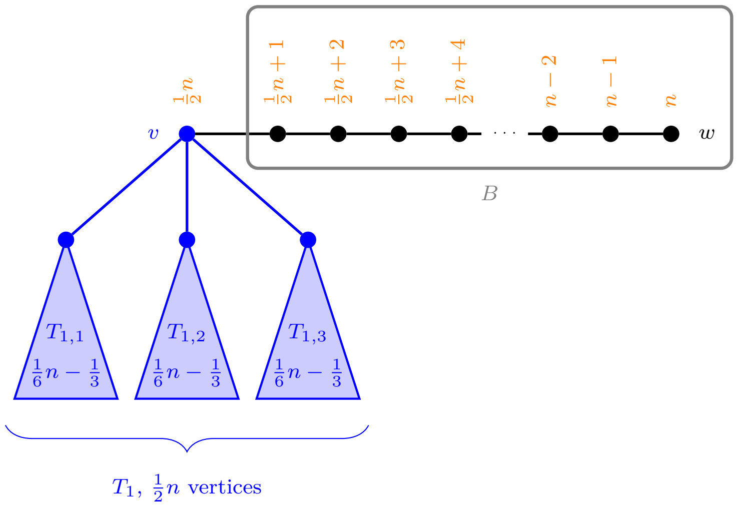
c) The unique minimum bisection (B, W). The vertices in B are colored blue, the vertices in W are colored green. One edge is cut and colored red.

Figure 6.1: Example of a bounded-degree tree with a 4-section of width 6, where the recursive approach yields a 4-section of much larger width.

Next, we have a closer look why a k-section that is recursively constructed with Theorem 1.1 can cut much more edges than a minimum k-section. Theorem 1.1 promises that every bounded-degree tree with linear diameter allows a bisection (B, W) of constant width. However, nothing is known about the diameters of the graphs induced by the sets B and W. This is illustrated by the next example, which is similar to the example above, but easier for applying the algorithm in Theorem 5.14, the algorithmic version of Theorem 1.1. As in the previous example, let $n \geq 4$ be an arbitrary integer such that there is a perfect ternary tree T_1 on $n_1 := \frac{1}{2}n$ vertices with root v and $\frac{1}{2}n$ is even. Again, denote by $T_{1,1}$, $T_{1,2}$, and $T_{1,3}$ the isomorphic subtrees of T_1 that are rooted in the three children of v and define $n_{i,1} := \frac{1}{3}\left(\frac{1}{2}n - 1\right) = \frac{1}{6}n - \frac{1}{3}$, which is the number of vertices in $T_{1,i}$ for $i \in [3]$. Furthermore, attach a path P on $\frac{1}{2}n$ new vertices to v to obtain a tree T on n vertices. It is easy to see that T allows a 4-section of width 5, where the trees $T_{1,i}$ for $i \in [3]$ are in different sets of the 4-section, see Figure 6.2a). When the algorithm contained in Theorem 5.14 is applied to T, it first searches for a longest path $\tilde{P}$ in T, which will obviously contain every vertex from P. Then, it computes a $\tilde{P}$-labeling of T, which can look like this: v receives some label i and the vertices in P receive labels $i+1, \ldots, i+\frac{1}{2}n$, see Figure 6.2b). Recall that numbers deviating by a multiple of n from a label in $[n]$ were considered to be the same as that label. Next, the algorithm checks if there is a vertex $x \in V(\tilde{P})$ with $N_{\frac{1}{2}n}(x) \in V(\tilde{P})$, where $N_{\frac{1}{2}n}(x) = x + \frac{1}{2}n$, and indeed, v is such


$\frac{1}{12}n+\frac{1}{3}$
$\frac{1}{12}n+\frac{1}{3}$
$\frac{1}{12}n+\frac{1}{3}$
$\frac{1}{4}n$
v
$T_{1,1}$
$\frac{1}{6}n-\frac{1}{3}$
$T_{1,2}$
$\frac{1}{6}n-\frac{1}{3}$
$T_{1,3}$
$\frac{1}{6}n-\frac{1}{3}$

a) Subtrees are colored in orange, blue, violet, and green such that each color indicates one set of a 4-section of width 5. Edges colored in red are cut.



b) The blue subtrees are perfect ternary trees. The orange numbers are an example for a P-labeling, where P is a longest path in the tree.

Figure 6.2: Example of a bounded-degree tree with linear diameter that allows a 4-section of width 5 and where recursive bisections with Theorem 1.1 produce a 4-section of much larger width.

a vertex, since v received label i and the leaf w of P, which is not adjacent to v in T, received label $i+\frac{1}{2}n$. So, one possibility for the set B of the computed bisection is

$$B := \left\{v \in V(T)\colon \text{ the label of } v \text{ is between } i+1 \text{ and } i+\tfrac{1}{2}n\right\},$$

which is exactly the set of vertices of P. Now, in the next round of recursively constructing a 4-section in T, one has to find a bisection in $T[B] = T_1$, which will cut at least $\Omega(\log_2 n)$ edges, due to Theorem 2.4 as T_1 is a perfect ternary tree.

Furthermore, both examples can be extended to k-sections, where k is a power of 2. For 8-sections two copies of the tree T in the example for 4-sections are joined by an edge to obtain a tree T', which allows an 8-section of constant width but the recursive approach can find an 8-section which cuts $\Omega(\log n)$ edges. Two copies of T' can then be joined by an edge to obtain an example for a 16-section where the recursive approach is bad and so on. Once k is not considered to be constant, these examples become less interesting as the gap between the width of a minimum k-section and a recursively constructed k-section decreases. This is not a flaw of the above examples, but a natural effect. As any k-section in a tree cuts at least $k-1$ edges, a minimum k-section with $k = \Omega(n)$ in a tree cuts a linear fraction of all edges.

The problem with the 4-section in the second example is, that nothing is known about the diameter in the two subgraphs that are produced by the bisection applied to the input tree. More precisely, it shows that the construction behind Theorem 1.1 can produce a bisection (B, W) in a bounded-degree tree T with linear diameter, where $T[B]$ does not have linear diameter. Indeed, $T[B]$ has diameter $\mathcal{O}(\log n)$ here. Then, Theorem 1.1 does not promise a bisection of constant width in $T[B]$ anymore. Consequently, constructing a 4-section in a bounded-degree tree with linear diameter recursively with Theorem 1.1 results in an upper bound of $\Omega\left(\frac{n}{\log n}\right)$ for the width of the 4-section, but Theorem 1.15 promises a 4-section of width $\mathcal{O}(1)$.

Recall the definition of the relative diameter, that was used in Section 5.1 and Section 5.2 to compare the diameters of trees with different numbers of vertices and is also defined for forests. The *relative diameter* of a forest G on n vertices is

$$\operatorname{diam}^*(G) \ := \ \frac{1}{n} \sum_{i \in [\ell]} (\operatorname{diam}(G_i) + 1),$$

where G_1, G_2, …, G_ℓ denote the components of G. Observe that $0 < \operatorname{diam}^*(G) \leq 1$ for every forest G and, if G is a tree, then $\operatorname{diam}^*(G)$ denotes the fraction of vertices of G on a longest path in G. To avoid the difficulties mentioned in the previous paragraph, one could first pre-partition the input tree into $k-1$ pieces, which each receive the same amount of vertices of a fixed longest path. More precisely, consider a tree T on n vertices and assume that n is divisible by 4. Let P be a longest path in T and define $d := \operatorname{diam}^*(T)$. It is easy to partition $V(T)$ into three pieces $V_1, V_2, V_3 \subseteq V(T)$ such that each piece contains $\frac{1}{3}$ of the vertices of P and with $e_T(V_1, V_2, V_3) = 2$ as long as one does not care about the sizes of V_i for all $i \in [3]$. Then, each set V_i with $i \in [3]$ contains a path of length at least $\frac{1}{3}|V(P)| = \frac{1}{3}dn$ and one could search for a set $B_i \subseteq V_i$ with $|B_i| = \frac{1}{4}n$ for each $i \in [3]$. The set B_4 for the 4-section could then be formed by the remaining vertices. The problem is that there could be a set V_i with $|V_i| < \frac{1}{4}n$. However, the pigeon-hole principle promises that there is at least one set V_i with $|V_i| \geq \frac{1}{3}n \geq \frac{1}{4}n$. Without loss of generality assume that $|V_1| \geq |V_2| \geq |V_3|$. So one can first cut off a piece $B_1 \subseteq V_1$ with $|B_1| = \frac{1}{4}n$ and then take the remaining vertices $V_1 \setminus B_1$ and the set V_2, which together contain more than $\frac{1}{4}n$ vertices, and cut off a set B_2 of size $\frac{1}{4}n$. Then, one can take all vertices that are neither in B_1 nor in B_2 and compute a bisection (B_3, B_4), which results in a 4-section (B_1, B_2, B_3, B_4) in T. As each of these three cuts is in a forest with a path of length at least $\frac{1}{3}dn$ and, hence, relative diameter at least $\frac{1}{3}d$, Theorem 5.6 implies

$$e_T(B_1, B_2, B_3, B_4) \ \leq \ 3 + 3 \cdot \frac{8\Delta(T)}{\frac{1}{3}d} \ = \ 3 + \frac{72\Delta(T)}{d}.$$

Consequently, this approach produces a 4-section of constant width in bounded-degree trees with linear diameter. It can easily be generalized for larger values of k, including values of k that are not powers of 2. However, this does not produce a k-section of the desired width for general k. Indeed, consider the tree T in Figure 6.2 and denote by $\tilde{P}$ a longest path in T. When cutting T into $k-1$ pieces such that each piece V_ℓ contains $\frac{1}{k-1} \cdot |V(\tilde{P})|$ vertices of the path $\tilde{P}$, the largest piece contains the entire tree T_1, i. e., at least $\frac{1}{2}n$ vertices (assuming that $\frac{1}{k-1} \cdot |V(\tilde{P})|$ is larger than the height of T_1). For $\ell \in [k-1]$, denote by $\tilde{V}_\ell$ the vertex set, such that $T[\tilde{V}_\ell]$ is the subgraph to which Theorem 5.6 is applied for the ℓth time. Then, for $\ell \leq \frac{k}{4}$, we have $|\tilde{V}_\ell| \geq \frac{n}{4}$ as the remaining vertices of T_1 are passed to the next set and each set B_ℓ contains at most $\frac{n}{k}$ vertices of T_1. Thus, for $\ell \leq \frac{k}{4}$, the best estimate on the relative diameter of $T[\tilde{V}_\ell]$ is

$$\frac{\frac{1}{k-1}|V(\tilde{P})|}{|\tilde{V}_\ell|} \ \leq \ \frac{\frac{1}{k-1} \cdot |V(\tilde{P})|}{\frac{1}{4}n} \ = \ \frac{4}{k-1} \operatorname{diam}^*(T).$$

Hence, the bound obtained for the k-section in T is at least

$$\frac{k}{4} \cdot \frac{8\Delta(T)}{\frac{4}{k-1}\operatorname{diam}^*(T)} \;\approx\; \tfrac{1}{2}k^2 \frac{\Delta(T)}{\operatorname{diam}^*(T)},$$

which grows with k^2. However, the aim for trees is to show a bound that only grows with k. The problem of this method is, that the first pieces $\tilde{V}_\ell$ might be rather large. As this example shows, they can contain almost all vertices but they contain only $\frac{1}{k-1}$ of the vertices of the path, i. e., the relative diameter of $T[V_1]$ could be around $\frac{1}{k-1}\operatorname{diam}^*(T)$. In the next section, we present a method that chooses the pieces $\tilde{V}_\ell$ more carefully, such that $\operatorname{diam}^*(T[\tilde{V}_\ell]) \geq \frac{1}{2}\operatorname{diam}^*(T)$, which then results in the desired bound for the width of a minimum k-section in a bounded-degree tree with linear diameter.

6.2 Minimum k-Section in Trees

The aim of this section is to prove Theorem 1.15, which provides an upper bound for the width of a minimum k-section in trees. In particular, Theorem 1.15 promises that every bounded-degree tree with linear diameter admits a k-section of width $\mathcal{O}(k)$ for every $k \geq 2$. First, Theorem 1.15 is restated and generalized to cutting a forest into k pieces of specified sizes. Using that $\frac{1}{\operatorname{diam}^*(T)} \leq \frac{n}{\operatorname{diam}(T)}$ holds for every tree T on n vertices, Theorem 1.15 follows easily from the next theorem.

Theorem 6.1 (Theorem 1.15 restated and generalized).
For every $k \geq 2$, for every forest G on n vertices, and for all $m_1, m_2, \dots, m_k \in \mathbb{N}_0$ with $\sum_{\ell=1}^{k} m_\ell = n$, there is a cut $(B_1, B_2, \dots, B_k)$ in G with $|B_\ell| = m_\ell$ for all $\ell \in [k]$ and

$$e_G(B_1, B_2, \dots, B_k) \;\leq\; (k-1) \cdot \left(2 + \frac{16}{\operatorname{diam}^*(G)}\right) \Delta(G).$$

A cut $(B_1, B_2, \dots, B_k)$ with these properties can be computed in $\mathcal{O}(kn)$ time.

As mentioned in the previous section, constructing a k-section by bisecting the graph repeatedly can give bad results and, hence, we follow a different approach: The main idea is to cut off one set for the k-section at a time while ensuring that the relative diameter of the remaining forest does not decrease, which is stated formally in Lemma 6.2 below. The lemma looks similar to Theorem 1.1 or its version for m-cuts, which was stated in Theorem 5.6. The main difference is that it contains additional information on the relative diameter of the subgraph induced by the white set of the cut. As this lemma does not follow from a refined analysis of the construction used in the proof of Theorem 5.6, it becomes unavoidable that the bound on the width of the cut increases.

Lemma 6.2.
For every forest G on n vertices and for every $m \in [n-1]$, there is an m-cut (B, W) in G that satisfies $\operatorname{diam}^*(G[W]) \geq \operatorname{diam}^*(G)$ *and*

$$e_G(B, W) \leq \left(2 + \frac{16}{\operatorname{diam}^*(G)}\right) \Delta(G).$$

A cut (B, W) with these properties can be computed in $\mathcal{O}(n)$ time.

Before proving Lemma 6.2 it is shown that Theorem 6.1 follows from Lemma 6.2 by applying Lemma 6.2 for $k-1$ times to cut off m_ℓ vertices in the ℓth application.

Proof of Theorem 6.1. Fix an integer $k \geq 2$. Let G be a forest on n vertices and let $m_1, \dots, m_k$ be as in the statement. For every $\ell \in [k]$ with $m_\ell = 0$, define $B_\ell = \emptyset$. Recall that $[0] := \emptyset$. For every $\ell \in [k-1]$ with $m_\ell \geq 1$, apply Lemma 6.2 to $G_\ell := G - \left(\bigcup_{h \in [\ell-1]} B_h\right)$ with size-parameter $m = m_\ell$ to obtain a set $B_\ell \subseteq V(G_\ell)$ with $|B_\ell| = m_\ell$. Then, exactly m_k vertices remain and defining B_k to be the set of these remaining vertices gives a cut $(B_1, \dots, B_k)$, where each set has the desired size. Note that, for every $\ell \in [k-1]$, the relative diameter of the forest G_ℓ satisfies $\operatorname{diam}^*(G_\ell) \geq \operatorname{diam}^*(G)$ and the maximum degree of G_ℓ is at most $\Delta(G)$. Consequently, at most $\left(2 + \frac{16}{\operatorname{diam}^*(G)}\right)\Delta(G)$ edges are cut in each of the at most $k-1$ applications of Lemma 6.2, which gives the desired bound. As for each $\ell \in [k-1]$, the computation of B_ℓ and G_ℓ takes $\mathcal{O}(n)$ time by Lemma 6.2 and Corollary 2.23, the cut can be computed in $\mathcal{O}(kn)$ time. □

Next, the proof of Lemma 6.2 is presented. As indicated by the example in Figure 6.2, a refined analysis of Theorem 1.1 will not work. Still, Theorem 1.1 will be used as a tool in the proof of Lemma 6.2. To avoid the problems described in Section 6.1, i. e., to ensure that the relative diameter of the graph induced by the white set is at least as large as in the original forest, Theorem 1.1 will be applied to a carefully chosen subgraph $\tilde{G} \subseteq G$. On the one hand, $\tilde{G}$ needs to have large relative diameter such that the bound in Theorem 1.1 is low when applied to $\tilde{G}$. On the other hand, the relative diameter of the graph induced by the white set of the computed cut will roughly be the relative diameter of $G - V(\tilde{G})$, so $G - V(\tilde{G})$ needs to have a large relative diameter. Note that these two conditions compete against each other, as to satisfy them many vertices of a longest path in G need to go to $\tilde{G}$ or $G - V(\tilde{G})$, respectively. The main part of the proof of Lemma 6.2 is to construct a subgraph $\tilde{G}$ with these properties. To do so, some notions from the proof of Lemma 5.7, which was the main part of the proof of Theorem 5.6, will be used again and are repeated now. For more details see Page 151. Consider a tree $G = (V, E)$ on n vertices and a path $P = (V_P, E_P) \subseteq G$ with ends x_0 and y_0. When removing all edges in E_P from G, then G decomposes into trees, one tree T_v for every $v \in V_P$. For each $v \in V_P$, let $T'_v := V(T_v) \setminus \{v\}$. For $x \in V(G)$, the unique vertex $v \in V_P$ with $x \in V(T_v)$ is called the *path-vertex* of x. A *P-labeling* of G is a labeling of the vertices of G with $1, 2, \dots, n$ such that the following holds:

- For each $v \in V_P$, the vertices of T_v have consecutive labels and v has the largest label among all vertices in T_v.
- For all $v, v' \in V_P$ with $v \neq v'$, if x_0 is closer to v than to v', then the label of v is smaller than the label of v'.

Again, each vertex will be identified with its label and any number differing by a multiple of n from a label in $[n]$ is considered to be the same as this label. When talking about labels and vertices, in particular when comparing them, we always refer to the integer in $[n]$. For three vertices $a, b, c \in V$ with $a \neq c$, we say that *b is between a and c* if $b = a$, $b = c$, or if starting at a and going along the numeration given by the labeling reaches b before c. If $a = c$, then we say that b is between a and c if $b = a = c$. For technical reasons, we will refer to the pair $\{y_0, x_0\}$ as an edge of G, even if it is not. For a vertex $v' \in V_P$, the vertex $v \in V_P$ is the *vertex after v' on P* if the tree T_v contains the vertex $v' + 1$. Then, $\{v', v\}$ is called the *edge after v' on P*. Similarly, in this case, $\{v', v\}$ is called the *edge before v on P* and v' is called the *vertex before v on P*. Last but not least, recall Lemma 5.2a), which says that, for every forest G with relative diameter d and $\Delta(G) \geq 3$, there is a tree T with $G \subseteq T$ and $\operatorname{diam}^*(T) = d$, and which allows us to work with trees instead of forests.

Proof of Lemma 6.2. Let $G = (V, E)$ be a forest on n vertices and fix some $m \in [n-1]$. If $\Delta(G) \leq 2$ then G is a collection of paths and so is every subgraph of G. Hence, G itself and every subgraph of G has relative diameter 1 and the m-cut in Lemma 5.2b) satisfies all the requirements for Lemma 6.2.

So from now on, assume that $\Delta(G) \geq 3$. By Lemma 5.2a) we may assume without loss of generality that G is connected. Let $P = (V_P, E_P)$ be a longest path in G and denote by x_0 and y_0 the ends of P. Set $d := \operatorname{diam}^*(G)$, and note that $|V_P| = dn$. Fix a P-labeling of the vertices of G, where x_0 receives label 1 and y_0 receives label n, and identify each vertex with its label.

For two vertices $x, y \in V$, the *P-distance* of x and y is defined as

$$d_P(x, y) = |\{v \in V_P \colon\ v \text{ is between } x \text{ and } y,\ v \neq y\}|\,.$$

It is easy to see that for every $x, y \in V$ with $x \neq y$

$$d_P(x, y) - d_P(x+1, y) \ \in\ \{0, 1\}$$

and

$$d_P(x+1, y) - d_P(x+1, y+1) \ \in\ \{0, -1\}.$$

Therefore

$$|d_P(x, y) - d_P(x+1, y+1)| \ \leq\ 1 \qquad\qquad \text{for every } x, y \in V. \tag{6.1}$$

Note that, for every $x \in V$ and all integers $\ell, \ell' \in [n-1]$, we have $d_P(x, x) = 0$ and

$$d_P(x, x+\ell) + d_P(x+\ell, (x+\ell)+\ell') \ =\ \begin{cases} d_P(x, x+\ell+\ell') & \text{if } \ell+\ell' < n, \\ |V_P| + d_P(x, x+\ell+\ell') & \text{if } \ell+\ell' \geq n. \end{cases}$$

Recall that x_0 was defined to be an end of P. Now, define $x_\ell = x_0 + \ell m$ for all $\ell \in [n]$. Then, $x_n = x_0 + nm = x_0$ and

$$\sum_{\ell=1}^{n} d_P(x_{\ell-1}, x_\ell) \ =\ m|V_P| \ =\ mdn.$$

Therefore, there are two vertices $x', x'' \in V$ with

$$d_P(x', x'+m) \leq md \qquad \text{and} \qquad d_P(x'', x''+m) \geq md.$$

Using that $d_P(x, y)$ is an integer for all x, $y \in V$ to strengthen the first inequality and loosening the second one, it follows that

$$d_P(x', x'+m) \leq \lfloor md \rfloor \qquad \text{and} \qquad d_P(x'', x''+m) \geq \lfloor md \rfloor\,.$$

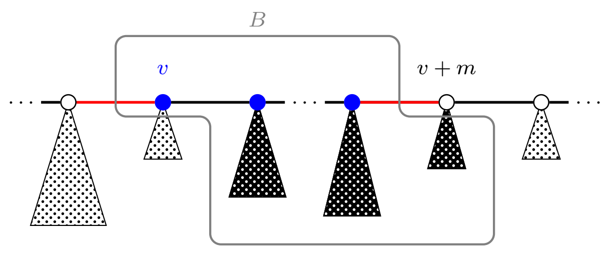
Now, (6.1) implies that there is a vertex $x^* \in V$ with $d_P(x^*, x^*+m) = \lfloor md \rfloor$. Let p_{x^*} and p_{x^*+m} be the path-vertices of x^* and x^*+m, respectively. Define $h := \min\{p_{x^*} - x^*, p_{x^*+m} - (x^*+m)\}$. Set $v := x^* + h$ and note that $v \in V_P$ or $v + m \in V_P$. Furthermore, x^* and v are in the same tree T_u with $u \in V_P$ and also x^*+m and $v+m$ are in the same tree $T_{u'}$ with $u' \in V_P$. As $v+m$ is not counted in $d_P(v, v+m)$, it follows that $d_P(v, v+m) = d_P(x^*, x^*+m) = \lfloor md \rfloor$. Define

$$M := \{u \in V \colon\ u \text{ is between } v \text{ and } v+m-1\}.$$

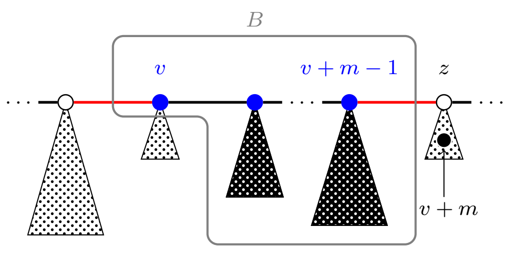
In the following figures, the path P will be drawn in the top and the trees T_u for $u \in V_P$ are drawn underneath P. The vertices in P that are counted in $d_P(v, v+m)$ will be colored blue and edges that are cut will be colored red, whenever they are drawn explicitly.

Case 1: $v \in V_P$ and $v+m \in V_P$.
Define $B := M$ and $W := V \setminus B$. Then the cut (B, W) cuts only edges that are incident to v or $v+m$, which implies $e_G(B, W) \leq 2\Delta(G)$, see Figure 6.3a). Furthermore, $|B| = m$ and

$$\operatorname{diam}^*(G[W]) \ \geq\ \frac{|V_P \cap W|}{|W|} \ =\ \frac{|V_P| - d_P(v, v+m)}{|W|} \ \geq\ \frac{dn - md}{n-m} \ =\ d.$$



a) Case 1, where $v \in V_P$ and $v+m \in V_P$.



b) Case 2a, where $v \in V_P$ and $v+m-1 \in V_P$.

Figure 6.3: Proof of Lemma 6.2. Construction of the black set in Case 1 and Case 2a.

Case 2: $v \in V_P$ and $v+m \notin V_P$.
Let z be the path-vertex of $v+m$. Observe that $z \notin M$ as otherwise $V_P \subseteq M$ and $\lfloor dm \rfloor = |V_P| = dn$, which contradicts $m \in [n-1]$. None of the edges in T_z is cut by $(M, V \setminus M)$ when $v+m-1 \in V_P$ and this case is treated separately for technical reasons.

Case 2a: $v+m-1 \in V_P$.
As in Case 1, the cut (B, W) with $B := M$ and $W := V \setminus M$ satisfies all requirements, see Figure 6.3b).

Case 2b: $v+m-1 \notin V_P$.
Observe that the cut $(M, V \setminus M)$ might cut too many edges in the tree T_z and the cut $(M \cup T'_z, V \setminus (M \cup T'_z))$ does not cut many edges, but applying Theorem 5.6 to $G[M \cup T'_z]$ might cut too many edges as its relative diameter can be much less than d, for example when T'_z contains $\Omega(n)$ vertices but m is small. So, instead of using $M \cup T'_z$, we will now define a set $\tilde{V} \subseteq M \cup T'_z$ such that $(\tilde{V}, V \setminus \tilde{V})$ cuts few edges, $\tilde{V}$ contains all vertices counted by $d_P(v, v+m)$, and $|\tilde{V}| \leq 2m$, which will ensure that $\operatorname{diam}^*(G[\tilde{V}]) \geq \frac{1}{2}d$.

Let $\tilde{m} = 2|T'_z \cap M|$, which satisfies $2 \leq \tilde{m} \leq 2m$ as $v+m-1$ is in T'_z due to the assumption of Case 2b. If $\tilde{m} \geq |T'_z|$, then define $B_z = T'_z$ and $W_z = \{z\}$, which satisfies $\frac{1}{2}\tilde{m} \leq |B_z| \leq \tilde{m}$ and $e_{T_z}(B_z, W_z) \leq \Delta(G)$. Otherwise, i. e., $\tilde{m} < |T'_z| < |V(T_z)|$, apply Lemma 4.1 to the tree T_z with parameter $\tilde{m}$ to obtain a cut (B_z, W_z) with $B_z \cup W_z = V(T_z)$, $\frac{1}{2}\tilde{m} \leq |B_z| \leq \tilde{m}$, and $e_{T_z}(B_z, W_z) \leq \Delta(G)$. Note that Remark 4.3 implies that $z \in W_z$. Define $\tilde{V} = (M \setminus T'_z) \cup B_z$ and note that $z \notin \tilde{V}$ as well as $|\tilde{V}| = m - \frac{1}{2}\tilde{m} + |B_z|$ and therefore $m \leq |\tilde{V}| \leq 2m$. The graph $\tilde{G} := G[\tilde{V}]$ consists of at least two components as there are no edges from $M \setminus T'_z$ to $B_z \subseteq T'_z$, see Figure 6.4. As each vertex in $V_P \cap \tilde{V}$ is in $M \setminus T'_z$, the components $\tilde{G}[M \setminus T'_z]$ contribute at least $|V_P \cap \tilde{V}| = d_P(v, v+m)$ to the numerator of $\operatorname{diam}^*(\tilde{G})$. As $|B_z| \geq \frac{1}{2}\tilde{m} \geq 1$, the

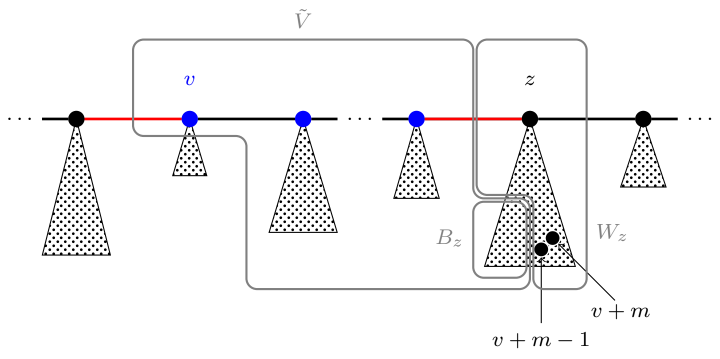


Figure 6.4: Proof of Lemma 6.2. Construction of $\tilde{V}$ in Case 2b, where $v \in V_P$, $v+m \notin V_P$, and $v+m-1 \notin V_P$. Note that $v+m-1$ and $v+m$ can also lie in B_z.

components of $G[B_z]$ contribute at least 1 to the numerator of $\operatorname{diam}^*(\tilde{G})$. Therefore,

$$\operatorname{diam}^*(\tilde{G}) \;\geq\; \frac{d_P(v, v+m)+1}{2m} \;\geq\; \frac{d}{2}.$$

Now, Theorem 5.6 applied to $\tilde{G}$ with size-parameter m implies that there is an m-cut $(\tilde{B}, \tilde{W})$ in $\tilde{G}$ with $e_{\tilde{G}}(\tilde{B}, \tilde{W}) \leq \frac{16}{d}\Delta(G)$. Define $B = \tilde{B}$ and $W = V \setminus B$. Every vertex from P that is not counted by $d_P(v, v+m)$ is in W by construction and therefore

$$\operatorname{diam}^*(G[W]) \cdot |W| \;\geq\; |W \cap V_P| \;\geq\; |V_P| - d_P(v, v+m) \;\geq\; dn - dm.$$

As $|W| = n - m$, it follows that $\operatorname{diam}^*(G[W]) \geq d$.

Next, the number of edges cut by (B, W) is estimated, which is at most the number of edges cut by $(\tilde{V}, V \setminus \tilde{V})$ plus the number of edges cut by $(\tilde{B}, \tilde{W})$ in $\tilde{G}$. The cut $(\tilde{V}, V \setminus \tilde{V})$ cuts at most $\Delta(G)$ edges within T_z as the cut (B_z, W_z) cuts at most $\Delta(G)$ edges in T_z. Recall that $z \in W_z \subseteq W$. Now, if v is the vertex before z on P, then $\tilde{V} = B_z \cup \{v\}$ and other than the edges in T_z only edges incident to v are cut by $(\tilde{V}, V \setminus \tilde{V})$, which results in $e_G(\tilde{V}, V \setminus \tilde{V}) \leq 2\Delta(G)$. So assume that the vertex after v on P is not z. Then, at most $\Delta(G) - 1$ edges incident to v are cut as v is either an end of P and therefore a leaf in G or the edge e after v on P exists and is not cut. Furthermore, from the edges incident to z that are cut by $(\tilde{V}, V \setminus \tilde{V})$, only the edge before z on P is not yet counted. Hence, in total at most $2\Delta(G)$ edges are cut by the cut $(\tilde{V}, V \setminus \tilde{V})$. Using that $(\tilde{B}, \tilde{W})$ cuts at most $\frac{16}{d}\Delta(G)$ edges in $\tilde{G} = G[\tilde{V}]$ gives the desired bound on the number of cut edges.

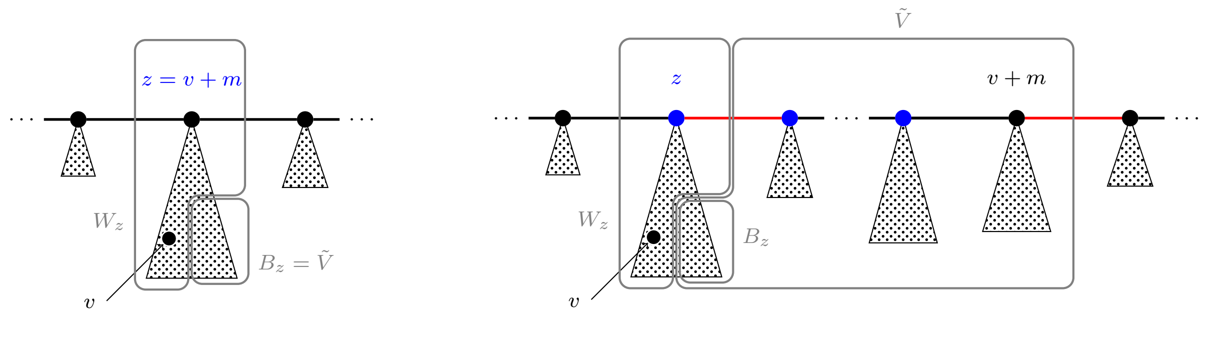
Case 3: $v \notin V_P$ and $v + m \in V_P$.
This case is similar to Case 2b, but some arguments need to be adjusted as the labeling cannot simply be reversed to obtain a labeling with the same properties due to the requirement that each $u \in V_P$ receives the largest label among all vertices in T_u. Denote by z the path-vertex of v. As in Case 2b, let $\tilde{m} = 2|T'_z \cap M|$ and note that $\tilde{m} \geq 2$ as $v \in T'_z$. If $\tilde{m} \geq |T'_z|$, let $B_z = T'_z$ and $W_z = \{z\}$. Otherwise apply Lemma 4.1 to T_z with size-parameter $\tilde{m}$ to obtain a cut (B_z, W_z) in T_z. Note that in both cases $z \in W_z$, $e_{T_z}(B_z, W_z) \leq \Delta(G)$, and $\frac{1}{2}\tilde{m} \leq |B_z| \leq \tilde{m}$. For technical reasons, the case when $z = v + m$ is treated separately.

Case 3a: $z = v + m$.
Then, $d_P(v, v+m) = 0$ and $md < 1$. Furthermore, $M \subseteq V(T_z)$ and $\tilde{m} = 2m$. Define $\tilde{V} := B_z$ and $\tilde{G} := G[\tilde{V}]$, which satisfy $m \leq |\tilde{V}| \leq 2m$ and $\operatorname{diam}^*(\tilde{G}) \geq \frac{1}{2m} \geq \frac{md}{2m} \geq \frac{1}{2}d$, see Figure 6.5a). Similarly to Case 2b, Theorem 5.6 says that there is an m-cut $(\tilde{B}, \tilde{W})$ in $\tilde{G}$ with $e_{\tilde{G}}(\tilde{B}, \tilde{W}) \leq \frac{16}{d}\Delta(G)$. Define $B = \tilde{B}$ and $W = V \setminus B$. Now, all vertices of P are in W, so $\operatorname{diam}^*(G[W]) \geq d$. Moreover, at most $\Delta(G)$ edges are cut by (B_z, W_z) in T_z and the cut $(\tilde{V}, V \setminus \tilde{V})$ cuts no other edges in G than the ones cut by (B_z, W_z) in T_z, as z is always in W_z. Using that the cut $(\tilde{B}, \tilde{W})$ cuts at most $\frac{16}{d}\Delta(G)$ edges in $\tilde{G}$, the desired bound is obtained.

Case 3b: $z \neq v + m$.
Define

$$\tilde{V} := (M \setminus (T'_z \cup \{z\})) \cup B_z \cup \{v + m\},$$

see Figure 6.5b). This definition of $\tilde{V}$ is slightly different than in Case 2b, as here $z \in M$ but $v + m$ is in $\tilde{V}$ instead of z, which will decrease the bound on the cut edges later. Now, $\tilde{V}$ contains exactly $d_P(v, v+m)$ vertices of P, which are all vertices counted in $d_P(v, v+m)$ except z, which is replaced by $v + m$. Recall that $z \neq v + m$ by the assumption of Case 3b. As $B_z \neq \emptyset$ and there are no edges between B_z and $\tilde{V} \setminus B_z$, one can argue similarly to Case 2b that $m \leq |\tilde{V}| \leq 2m$ as well as that $\tilde{G} := G[\tilde{V}]$ satisfies $\operatorname{diam}^*(\tilde{G}) \geq \frac{1}{2}d$. Then, Theorem 5.6 says that there is an m-cut $(\tilde{B}, \tilde{W})$ in $\tilde{G}$ with $e_{\tilde{G}}(\tilde{B}, \tilde{W}) \leq \frac{16}{d}\Delta(G)$. Define $B = \tilde{B}$



a) Case 3a, where $z = v + m$.

b) Case 3b, where $z \neq v + m$.

Figure 6.5: Proof of Lemma 6.2, construction of $\tilde{V}$ in Case 3, where $v \notin V_P$ and $v + m \in V_P$. Note that, in both cases, v can also lie in B_z.

and $W = V \setminus B$. As mentioned before, $d_P(v, v + m) = \lfloor md \rfloor$ vertices of P are in $\tilde{V}$. Therefore, at most $\lfloor md \rfloor$ vertices of P are in B and as in Case 2b, it follows that $\operatorname{diam}^*(G[W]) \geq d$. The number of cut edges can be estimated similarly to Case 2b: At most $\Delta(G)$ edges are cut by (B_z, W_z) in T_z and z is always in $V \setminus \tilde{V}$. Other than these edges, the cut $(\tilde{V}, V \setminus \tilde{V})$ cuts only the edge after z on P and the edge after $v + m$ on P. Due to the assumption $\Delta(G) \geq 3$, the width of the cut $(\tilde{V}, V \setminus \tilde{V})$ in G is at most $2\Delta(G)$. Using that the cut $(\tilde{B}, \tilde{W})$ cuts at most $\frac{16}{d}\Delta(G)$ edges in $\tilde{G}$, the desired bound is obtained.

Next, it is discussed how to implement an algorithm that computes a cut (B, W) in a forest G on n vertices with the desired properties. The construction presented above is summarized in Algorithm 6.1, which is only for trees. This suffices for the following reasons. Consider a forest G' that is not connected. If $\Delta(G') \leq 2$, then the cut in Lemma 5.2b) has the desired properties as argued above and can be computed in $\mathcal{O}(n)$ time by Lemma 5.16b). Otherwise, $\Delta(G') \geq 3$ and Lemma 5.16a) states that, in $\mathcal{O}(n)$ time, a tree T' containing the forest G' as a subgraph and such that $\Delta(T') = \Delta(G')$ and $\operatorname{diam}^*(T') = \operatorname{diam}^*(G)$ can be computed. Furthermore, in this case, every cut (B', W') in T' is also a cut in G' and $e_{G'}(B', W') \leq e_{T'}(B', W')$. Thus, it suffices to compute a cut in T'. So, from now on, assume that $G = (V, E)$ is a tree. Observe that Lines 12-13 treat Case 1 and Case 2a, Lines 15-18 treat Case 2b, and Lines 20-23 treat Case 3a and Case 3b, whose construction differs only in the definition of the set $\tilde{V}$.

As in the implementation of the algorithm contained in Theorem 5.6, Algorithm 6.1 does not identify vertices with their labels. From now on, $L(x)$ is used to refer to the label of a vertex $x \in V$ and $L^{-1}(a)$ is used to refer to the vertex which received label $a \in [n]$. For example, for a vertex $v \in V$, the vertex whose label comes m steps after the label of v in the numeration is $L^{-1}(L(v) + m)$. Furthermore, the definition of the P-distance is adjusted such that it refers to two vertex names, i. e.,

$$d_P(x, y) = |\{v \in V_P \colon\ L(v) \text{ is between } L(x) \text{ and } L(y),\ v \neq y\}|$$

for all $x, y \in V(G)$. As usually, only the set B of the desired cut (B, W) is returned and throughout the algorithm, all sets are stored as unordered lists of vertices.

Next, the running time of Algorithm 6.1 is analyzed line by line. The aim is to show that Algorithm 6.1 can be implemented to run in linear time, i. e., $\mathcal{O}(n)$ time. As there are no loops, it suffices to show that each line can be implemented to take at most linear time. Lines 1-3 take $\mathcal{O}(n)$ time by Lemma 5.16b). As in Algorithm 5.3 in the proof of Lemma 5.19, Lines 4-6 take $\mathcal{O}(n)$ time. Recall that there, it was argued that the P-labeling can be stored in two arrays, such that the algorithm can convert between labels and vertex names in constant time. Note that for each $x \in V$ the number $d_1(x)$ as defined in Line 7 is the number of vertices in V_P whose label is smaller than x. Hence, Line 7 takes $\mathcal{O}(n)$ time by

Algorithm 6.1: Computing an m-cut in a tree with the properties in Lemma 6.2.

```
   Input: tree $G = (V, E)$ on $n$ vertices and $m \in [n-1]$.
   Output: $m$-cut $(B, W)$ with the properties in Lemma 6.2.
 1 If $\Delta(G) \le 2$ then
 2     Return an arbitrary $m$-cut $(B, W)$ in $G$ with $e_G(B, W) \le \Delta(G)$;
 3 Endif
 4 Compute a longest path $P = (V_P, E_P)$ in $G$ and let $L_P$ be a list of the vertices on $P$;
 5 $d \leftarrow \mathrm{diam}^*(G)$;
 6 Compute a $P$-labeling of the vertices in $G$ and the path-vertex for every vertex of $G$.
       For each $x \in V$, denote by $L(x)$ the label of $x$ and, for each $a \in [n]$, denote by $L^{-1}(a)$ the
       vertex that received label $a$;
 7 For each $x \in V$ compute $d_1(x) = d_P(L^{-1}(1), x)$;
 8 Determine a vertex $v \in V$ with $d_P(v, L^{-1}(L(v) + m)) = \lfloor md \rfloor$ and $v \in V_P$ or $v + m \in V_P$;
 9 $M \leftarrow \{w \in V\colon\ L(w)$ is between $L(v)$ and $L(v) + m - 1\}$;
10 If $v \in V_P$ then
11     If $L^{-1}(L(v) + m) \in V_P$ or $L^{-1}(L(v) + m - 1) \in V_P$ then
12         $B \leftarrow M$;
13         Return $(B, V \setminus B)$;
14     Endif
15     Let $z$ be the path-vertex of $v + m$;
16     $\tilde{m} \leftarrow 2|T'_z \cap M|$;
17     Let $(B_z, W_z)$ be a cut in $T_z$ with $e_{T_z}(B_z, W_z) \le \Delta(G)$, $z \in W_z$, and $\frac{1}{2}\tilde{m} \le |B_z| \le \tilde{m}$;
18     $\tilde{V} \leftarrow (M \setminus T'_z) \cup B_z$;
19 Else
20     Let $z$ be the path-vertex of $v$;
21     $\tilde{m} \leftarrow 2|T'_z \cap M|$;
22     Let $(B_z, W_z)$ be a cut in $T_z$ with $e_{T_z}(B_z, W_z) \le \Delta(G)$, $z \in W_z$, and $\frac{1}{2}\tilde{m} \le |B_z| \le \tilde{m}$;
23     If $z = v + m$ then $\tilde{V} = B_z$ else $\tilde{V} \leftarrow (M \setminus (T'_z \cup \{z\})) \cup B_z \cup \{v + m\}$;
24 Endif
25 Compute an $m$-cut $(\tilde{B}, \tilde{W})$ in $G[\tilde{V}]$ with the properties in Theorem 5.6;
26 $B \leftarrow \tilde{B}$;
27 Return $(B, V \setminus B)$;
```

using the List L_P. Using $d_1(x)$, the algorithm can compute $d_P(x, x+m)$ and $d_P(x-m, x)$ for all $x \in V_P$ since $d_P(x, y) = d_1(y) - d_1(x)$ for all $x, y \in V$ with $L(x) \le L(y)$. As argued above, a vertex v with the properties in Line 8 always exists and can now be determined in $\mathcal{O}(|V_P|) = \mathcal{O}(n)$ time. The set M in Line 9 can be read off the labeling in $\mathcal{O}(n)$ time. Using that $x \in V_P$ if and only if the path-vertex of x is x itself, the algorithm can determine in constant time whether a vertex $x \in V$ is in V_P. Hence, Lines 10-11 take constant time and Lines 12-13, if executed, take $\mathcal{O}(n)$ time together. Assume for now that Lines 15-18 are executed. As the algorithm already computed all path-vertices, Line 15 takes constant time. Next, the algorithm determines the vertex z' on P before z by using the list L_P, which takes $\mathcal{O}(n)$ time. Note that $T'_z \cap M$ is precisely the set of vertices with labels between $L(z') + 1$ and $L(v) + m - 1$ and, hence, determining $\tilde{m}$ in Line 16 takes $\mathcal{O}(n)$ time. To implement Line 17, note that T'_z is the set of vertices with labels between $L(z') + 1$ and $L(z) - 1$, and can be read off the labeling. As described above, if $|T'_z| \leq \tilde{m}$, then the algorithm sets $B_z = T'_z$. Otherwise, it computes the tree T_z and applies the algorithm in Lemma 4.1 to T_z with size-parameter $\tilde{m}$, which together takes at most $\mathcal{O}(n)$ time. Moreover, $M \setminus T'_z$ is the set of vertices with labels between $L(v)$ and $L(z')$ and can be read off the labeling in $\mathcal{O}(n)$ time. Using that the sets $M \setminus T'_z$ and B_z are disjoint, Line 18 can be implemented as a concatenation of lists, and in total takes $\mathcal{O}(n)$ time. Similarly to Lines 15-18, one can argue that Lines 20-23 together take $\mathcal{O}(n)$ time. Finally, computing the subgraph $G[\tilde{V}]$ in Line 25 and applying the algorithm corresponding to Theorem 5.14 takes $\mathcal{O}(n)$ time, see also Remark 5.15. □

The proof of Lemma 6.2 uses Theorem 5.6 to estimate the width of the exact cut in $\tilde{G}$, which can be improved by using Theorem 5.12. Then, the bound on the width of the cut in Lemma 6.2 improves to

$$\begin{aligned} e_G(B,W) \;&\leq\; 2\Delta(G) + \frac{1}{2}\,\Delta(G)\left(\left(\log_2\left(\frac{2}{\operatorname{diam}^*(G)}\right)\right)^2 + 7\log_2\left(\frac{2}{\operatorname{diam}^*(G)}\right) + 6\right) \\ &\leq\; \frac{1}{2}\,\Delta(G)\left(\left(\log_2\left(\frac{1}{\operatorname{diam}^*(G)}\right)\right)^2 + 9\log_2\left(\frac{1}{\operatorname{diam}^*(G)}\right) + 18\right) \end{aligned} \tag{6.2}$$

and the following improved version of Theorem 6.1 is obtained.

Theorem 6.3 (improved version of Theorem 6.1).
For every $k \geq 2$, for every forest G on n vertices, and for all $m_1, m_2, \ldots, m_k \in \mathbb{N}_0$ with $\sum_{\ell=1}^{k} m_\ell = n$, there is a cut $(B_1, B_2, \ldots, B_k)$ in G with $|B_\ell| = m_\ell$ for all $\ell \in [k]$ and

$$e_G(B_1, B_2, \ldots, B_k) \;\leq\; (k-1)\,\frac{\Delta(G)}{2}\,\left(\left(\log_2\left(\frac{1}{\operatorname{diam}^*(G)}\right)\right)^2 + 9\log_2\left(\frac{1}{\operatorname{diam}^*(G)}\right) + 18\right).$$

A cut $(B_1, B_2, \ldots, B_k)$ with these properties can be computed in $\mathcal{O}(kn)$ time.

Last but not least for k-sections in trees, we mention that it looks like the analysis of the algorithm contained in Theorem 6.1 and Lemma 6.2 cannot easily be tightened to obtain an algorithm that computes a k-section in a tree in linear time such that the running time is independent of k. Indeed, fix an integer $k \geq 2$ and let T be a tree on n vertices. For $\ell \in [k]$, the ℓ^{th} application of Lemma 6.2 takes $\mathcal{O}(n_\ell)$ time, where n_ℓ denotes the number of vertices in the remaining graph after cutting off the sets $B_1, \ldots, B_{\ell-1}$ for the final k-section $(B_1, \ldots, B_k)$. In the proof of Theorem 6.1, the number n_ℓ was estimated by $n_\ell \leq n$. Using the exact values, i. e., $n_\ell = n - (\ell-1)\frac{n}{k} = (k-\ell+1)\frac{n}{k}$ when n is divisible by k, the running time of computing a k-section is still proportional to

$$\sum_{\ell \in [k-1]} n_\ell \;=\; \frac{n}{k}\sum_{\ell=1}^{k-1}(k-\ell+1) \;=\; \frac{n}{k}\sum_{h=2}^{k} h \;=\; \frac{n}{k}\left(\frac{1}{2}k(k+1)-1\right) \;\geq\; \frac{n}{k}\cdot\frac{k^2}{2} \;=\; \frac{1}{2}nk.$$

So to achieve a running time of $\mathcal{O}(n)$ for computing a k-section in T, the ℓ^{th} application of Lemma 6.2 may only take $\mathcal{O}\left(\frac{n}{k}\right)$ time. This is not easy for the following two reasons: First, Line 8 in Algorithm 6.1 might require the algorithm to go through the entire set V_P, which can have size $\Omega(n)$. Second, the cuts in Line 17 and Line 22 might require to go through the entire tree T_z. So, if T_z is large, say it has $\Omega(n)$ vertices, then $T'_z \setminus \tilde{V}$ is large and the algorithm might have to go through $T'_z \setminus \tilde{V}$ again if the same vertex z is chosen in the next application of Algorithm 6.1.

However, the running time can become linear in certain cases when a cut into k pieces of specified size is desired. For example, consider the case when $n_\ell = c^\ell \tilde{n}$ for all $\ell \in [k]$ with some suitable constants $\tilde{n} \in \mathbb{N}$

and $c \in [0,1)$, i. e., a cut into k pieces of sizes $m_\ell = n_\ell - n_{\ell+1} = (1-c)c^\ell \tilde{n}$ for $\ell \in [k-1]$ and $m_k = n_k$ is desired. Then, the running time is bounded by

$$\mathcal{O}\left(\sum_{\ell\in[k-1]} n_\ell\right) = \mathcal{O}\left(c\tilde{n}\sum_{\ell=0}^{k-2} c^\ell\right) = \mathcal{O}(c\tilde{n}) = \mathcal{O}(n_1) = \mathcal{O}(n),$$

which is linear in the input size and does not depend on k.

6.3 Extension to General Graphs

The aim of this section is to generalize the bound for the width of a minimum k-section in a tree to general graphs by using tree decompositions, similarly as Theorem 1.1 can be generalized to Theorem 1.3. The techniques of both generalizations are very similar. Recall that, instead of working with the relative diameter, the general version of the theorem for bisections uses the relative weight of a heaviest path in a given tree decomposition. Before restating the theorem for k-sections in arbitrary graphs, this definition is quickly repeated. Consider a tree decomposition $(T, \mathcal{X})$ of some graph G with $\mathcal{X} = (X^i)_{i\in V(T)}$ and a path $P \subseteq T$. The *weight* of P with respect to $\mathcal{X}$ is $w_\mathcal{X}(P) := \left|\bigcup_{i\in V(P)} X^i\right|$ and the *relative weight* of P with respect to $\mathcal{X}$ is $w^*_\mathcal{X}(P) = \frac{1}{n} w_\mathcal{X}(P)$, where n denotes the number of vertices of G. Furthermore, $r(T, \mathcal{X})$ was defined to be the *relative weight of a heaviest path* in T, i. e., $r(T, \mathcal{X}) = w^*_\mathcal{X}(P^*)$ where $P^* \subseteq T$ is a path with $w_\mathcal{X}(P^*) \geq w_\mathcal{X}(P)$ for all paths $P \subseteq T$. See Section 5.3 for more details and a discussion on the relationship between the relative diameter in trees and the relative weight of a heaviest path in a tree decomposition.

Theorem 6.4 (Theorem 1.16 restated and generalized).
For every graph G on n vertices, for every tree decomposition $(T, \mathcal{X})$ of G of width $t-1$, and for all $m_1, m_2, \ldots, m_k \in \mathbb{N}_0$ with $\sum_{\ell=1}^k m_\ell = n$, there is a cut $(B_1, B_2, \ldots, B_k)$ in G with $|B_\ell| = m_\ell$ for all $\ell \in [k]$ and

$$e_G(B_1, B_2, \ldots, B_k) \leq (k-1)\frac{t\Delta(G)}{2}\left(\left(\log_2\left(\frac{1}{r(T,\mathcal{X})}\right)\right)^2 + 11\log_2\left(\frac{1}{r(T,\mathcal{X})}\right) + 24\right).$$

If $V(G) = [n]$, a cut $(B_1, B_2, \ldots, B_k)$ with these properties can be computed in $\mathcal{O}(k\|(T, \mathcal{X})\|)$ time and requires only the tree decomposition $(T, \mathcal{X})$ as input.

The main idea of the proof of this theorem is similar to the proof of Theorem 6.1, i. e., the pieces B_ℓ are cut off successively from the graph G while ensuring that the relative weight of a heaviest path in the tree decomposition induced by the remaining part is at least as large as the relative weight of a heaviest path in the original tree decomposition. Lemma 6.5 below states this formally. Afterwards, Theorem 6.4 is derived from Lemma 6.5. As in the case of trees, almost all technical details are hidden in this lemma. Its proof is a generalization of the proof of Lemma 6.2 and uses the same techniques as the generalization of Theorem 1.1 to Theorem 1.3 or, more precisely, the generalization of Lemma 5.7 to Lemma 5.22. Therefore, some notations and definitions from Section 5.3.2 are repeated before the proof of Lemma 6.5 is presented.

Proof of Theorem 1.16

As mentioned above, the main idea to prove Theorem 1.16 and its more general version Theorem 6.4 is the following lemma, which is a generalization of Lemma 6.2.

Lemma 6.5.
For every graph G on n vertices, for every tree decomposition $(T, \mathcal{X})$ of G of width $t-1$, and for every $m \in [n-1]$, there is an m-cut (B, W) in G with

$$e_G(B,W) \le \frac{t\Delta(G)}{2}\left(\left(\log_2\left(\frac{1}{r(T,\mathcal{X})}\right)\right)^2 + 11\log_2\left(\frac{1}{r(T,\mathcal{X})}\right) + 24\right),$$

and such that the tree decomposition $(T', \mathcal{X}')$ induced by $G[W]$ in $(T, \mathcal{X})$ satisfies $r(T', \mathcal{X}') \ge r(T, \mathcal{X})$. Let G_0 be an arbitrary graph with $G \subseteq G_0$ and $V(G_0) = [n_0]$ for some integer n_0 and let $(T_0, \mathcal{X}_0)$ be an arbitrary tree decomposition of G_0 such that $(T, \mathcal{X})$ is the tree decomposition induced by G in $(T_0, \mathcal{X}_0)$. If $(T_0, \mathcal{X}_0)$ and a list of the vertices in G are provided as input, then a cut (B, W) in G with the above properties can be computed in $\mathcal{O}(\|(T_0, \mathcal{X}_0)\|)$ time.

Proof of Theorem 6.4. Let G be an arbitrary graph, fix an arbitrary tree decomposition $(T, \mathcal{X})$ of G of width $t-1$, and let $m_1, \ldots, m_k$ be as in the statement. Without loss of generality, we may assume that $m_\ell > 0$ for all $\ell \in [k]$, as otherwise, for each $\ell \in [k]$ with $m_\ell = 0$, one can set $B_\ell = \emptyset$ in the beginning and discard m_ℓ. The idea is to apply Lemma 6.5 iteratively for $k-1$ times to obtain the sets $B_1, \ldots, B_{k-1}$. Define $G_0 = G$ and $(T_0, \mathcal{X}_0) = (T, \mathcal{X})$. For each $\ell \in [k-2]$, let $G_\ell = G - \bigcup_{h=1}^{\ell} B_h$ and define $(T_\ell, \mathcal{X}_\ell)$ as the tree decomposition that is induced by G_ℓ in $(T, \mathcal{X})$. Furthermore, for $\ell \in [k-1]$, denote by (B_ℓ, W_ℓ) a cut with the properties in Lemma 6.5 when applied to the graph $G_{\ell-1}$ with the tree decomposition $(T_{\ell-1}, \mathcal{X}_{\ell-1})$ and size-parameter m_ℓ. The application of Lemma 6.5 is always feasible as $m_k > 0$ by our assumption and, hence, $|V(G_{\ell-1})| \ge m_\ell + m_k > m_\ell$ for each $\ell \in [k-1]$. Finally, set $B_k = W_{k-1}$. Observe that, for all $\ell \in [k-1]$, the graph G_ℓ is identical to the graph $G_{\ell-1}[W_\ell]$ and $(T_\ell, \mathcal{X}_\ell)$ is also the tree decomposition induced by G_ℓ in $(T_{\ell-1}, \mathcal{X}_{\ell-1})$. So, $(B_1, \ldots, B_k)$ is a cut in G and Lemma 6.5 implies that the following invariants hold for each $\ell \in [k-1]$.

(i) The width of $(T_{\ell-1}, \mathcal{X}_{\ell-1})$ is at most $t-1$ and $r(T_{\ell-1}, \mathcal{X}_{\ell-1}) \ge r(T, \mathcal{X})$.

(ii) The cut (B_ℓ, W_ℓ) in the graph $G_{\ell-1}$ satisfies

$$e_{G_{\ell-1}}(B_\ell, W_\ell) \;\le\; \frac{t\Delta(G)}{2}\left(\left(\log_2\left(\frac{1}{r(T,\mathcal{X})}\right)\right)^2 + 11\log_2\left(\frac{1}{r(T,\mathcal{X})}\right) + 24\right).$$

Now, (ii) implies that the desired bound on the width of the cut $(B_1, B_2, \ldots, B_k)$ is satisfied. Moreover, the running time of the corresponding algorithm follows immediately from Lemma 6.5 by keeping track of the vertices in the graph G_ℓ for each $\ell \in [k-1]$. Observe that Proposition 2.20 cannot be applied as $V(G_{\ell-1}) = [n_{\ell-1}]$ for some integer $n_{\ell-1}$ is not necessarily satisfied. This can be accomplished through a binary array of length n_0, where the entries of the vertices not yet assigned to one of the sets $B_1, \ldots, B_{\ell-1}$ are set to one. Then, a list of the vertices in the remaining graph $G_{\ell-1} = G - \bigcup_{h=1}^{\ell-1} B_h$, that is used in the ℓ^{th} application of Lemma 6.5, can be obtained in $\mathcal{O}(n_0) = \mathcal{O}(\|(T, \mathcal{X})\|)$ time. □

The statement of the running time in Lemma 6.5 might look unnecessary complicated due to the extra graph G_0. Clearly, one can set $G = G_0$ and the running time of the algorithm contained in Lemma 6.5 becomes proportional to $\|(T, \mathcal{X})\|$. The proof of Theorem 6.4 shows that the graph G_0 and the tree decomposition $(T_0, \mathcal{X}_0)$ are convenient, as Lemma 6.5 is applied to subgraphs of the graph in which we would like to find a k-section. If Lemma 6.5 was stated with $G_0 = G$, then it would be necessary to return a tree decomposition $(T', \mathcal{X}')$ of G', which is not difficult due to Proposition 2.31. However, for reapplying Lemma 6.5, it then is necessary to set up a bijection between the vertices of G' and $[n']$, where n' denotes the number of vertices of G'. This might speed up the actual running time of the algorithm in Theorem 1.16, but does not speed up the asymptotic running time of $\mathcal{O}(k\|(T_0, \mathcal{X}_0)\|)$ for computing a k-section in the graph G_0 with the tree decomposition $(T_0, \mathcal{X}_0)$. Recall that, for trees, our method achieved a running time of $\mathcal{O}(kn)$ for computing a k-section in a tree on n vertices, see also Theorem 6.1. Furthermore, during the proof of Lemma 6.5 it will become clear that some parts of the original tree decomposition $(T_0, \mathcal{X}_0)$ might need to be traversed several times, for example when applying the simple approximate cuts or when computing a heaviest path.

Notation and Vertex Labeling

Here, some notations and the vertex labeling introduced in Section 5.3.2 are repeated. Readers familiar with these can skip this paragraph, as nothing new is introduced. Consider a tree decomposition $(T, \mathcal{X})$ of width $t-1$ and a path $P \subseteq T$, denote by $G = (V, E)$ its underlying graph, and let n be the number of vertices of G. Let $T = (V_T, E_T)$, $P = (V_P, E_P)$, $\mathcal{X} = (X^i)_{i \in V_T}$, and fix one end i_0 of P. Then, $R := \bigcup_{i \in V_P} X^i$ is called the *set of root vertices of* P and $S := V \setminus R$. To partition the sets S and R along the path P, denote by T_i the component of $T - E_P$ that contains i, for each $i \in V_P$. For each $x \in R$, the unique node $i \in V_P$ that is closest to i_0 among all nodes $j \in V_P$ with $x \in X^j$ is called the *path-node* of x. Then, defining

$$R_i := \{x \in X^i \colon\ i \text{ is the path-node of } x\} \qquad \text{and} \qquad S_i := \bigcup_{j \in V(T_i)} X^j \setminus R$$

for all $i \in V_P$ partitions the sets R and S according to Proposition 5.25b). Furthermore, for $x \in S$, the node $i \in V_P$ is called the *path-node* of x if and only if $x \in S_i$. Note that every vertex $x \in V$ has a unique path-node $i \in V_P$.

Consider two neighboring nodes i and j on P. We say that i is the *node before j on P*, if i is passed before j when traversing P from i_0 to its other end j_0, and otherwise we say that i is the *node after j on P*. For technical reasons, this notion is extended to the nodes i_0 and j_0 by saying that i_0 is the node after j_0 on P and that j_0 is the node before i_0 on P. Furthermore, the end i_0 of P is called *nonredundant* if $X^{i_0} \neq \emptyset$ and, if $i_0 \neq j_0$, additionally $X^i \not\subseteq X^{i^-}$ for all $i \in V_P \setminus \{i_0\}$, where i^- denotes the node before i on P. It is easy to see that every path $P \subseteq T$ is nonredundant, if $(T, \mathcal{X})$ is a nonredundant tree decomposition. Recall Proposition 5.25a), which says that, for every $i \in V_P$, the set R_i is not empty, if the path P is nonredundant.

When labeling the vertices of G with $\{1, 2, \ldots, n\}$, we say that the vertices in $V' \subseteq V(G)$ receive *consecutive labels* if there are $k, k' \in [n] \cup \{0\}$ such that each vertex in V' receives a label in $[k] \setminus [k']$ and vice versa. A *P-labeling* of G with respect to $(T, \mathcal{X})$ is a labeling of the vertices in V with $\{1, 2, \ldots, n\}$, such that

- for each node $i \in V_P$, the vertices of $R_i \cup S_i$ receive consecutive labels and the vertices in R_i receive the largest labels among those, and
- for all nodes $i, j \in V_P$ with $i \neq j$, if i_0 is closer to i than to j, then each vertex in $R_i \cup S_i$ has a smaller label than every vertex in $R_j \cup S_j$.

As mentioned above, the sets R_i and S_i form a partition of V and, hence, a P-labeling exists always. A P-labeling is useful for finding certain cuts, related to the sets R_i and S_i in the graph G. Fix an arbitrary node $i \in V_P \setminus \{i_0, j_0\}$. Then, the graph $G - E_G(i)$ decomposes into the following disjoint parts: an isolated vertex for every $v \in R_i$, the part $G[S_i]$, the part induced by the vertices in $\bigcup_{j \in V(P^-)}(R_j \cup S_j)$, and the part induced by the vertices in $\bigcup_{j \in V(P^+)}(R_j \cup S_j)$, where P^- and P^+ denote the paths into which P decomposes when i is removed. See Proposition 5.26 for the details. In the next section, we will again identify vertices with their labels for the existence part of the proof, and we will consider any number that differs by a multiple of n from a label in $[n]$ to be the same as that label. When we talk about labels and vertices, in particular when comparing them, we always refer to an integer in $[n]$. The notion of "a is between b and c" for three vertices $a, b, c \in V$ is adapted from Section 6.2 and was also used in Section 5.3.2.

The algorithm in Lemma 6.5, uses some subroutines from Section 5.3. Recall that most of these subroutines receive a tree decomposition $(T, \mathcal{X})$ as input that satisfies the restriction $V(G) = [n]$ for the underlying graph G, where $n := |V(G)|$. Hence, it will sometimes be necessary to set up bijections between the vertex set of the considered graph G' and $[n']$ for $n' := |V(G')|$. Lemma 5.32 and Lemma 5.35 say that a heaviest path P in $(T, \mathcal{X})$ and the set of P-parameters can be computed in $\mathcal{O}(\|(T, \mathcal{X})\|)$ time when the underlying graph G satisfies $V(G) = [n]$ for some integer n. In this section, bijections will be set up such that we can assume that $V(G) = [n]$. Then the set of P-parameters consists of

- a P-labeling of the vertices of G, stored in two integer arrays A_L and A_V, each of length n, such that, for $x \in V(G) = [n]$, the entry $A_L[x]$ is the label of vertex x and, for $\ell \in [n]$, the entry $A_V[\ell]$ is the vertex that received label ℓ,
- a binary array A_R of length n such that, for each $x \in V(G)$, the entry $A_R[x]$ is one if and only if $x \in R$,
- an integer array A_P of length n such that, for each $x \in V(G)$, the entry $A_P[x]$ is the path-node of x,
- the trees T_i for all i in P, each stored as an arborescence with root i, including pointers to the corresponding clusters from $(T, \mathcal{X})$, as well as,
- a list L_P of the nodes on P in the order in which they occur when traversing P from i_0 to j_0, including, for each $i \in V(P)$, a pointer to the root of T_i.

This is slightly less general than the original version introduced in Definition 5.33. Recall that all array indices refer to vertex names and not labels, except the ones of A_V. As in Section 5.3.4 and Section 5.3.5, the set of P-parameters is useful to store the P-labeling of the vertices and related information as the path-nodes. Furthermore, the set of P-parameters allows the algorithm to check in constant time whether a vertex x is in the set S_i for some fixed $i \in V_P$ as stated in Proposition 5.34b).

Proof of Lemma 6.5

The idea for the proof of Lemma 6.5 is very similar to its tree version, namely the proof of Lemma 6.2, see also Table 6.1 for an overview of the notation. One difference is that, instead of cutting along single edges of the decomposition tree, we will work with cuts related to removing entire clusters from the graph. This is also reflected by the slightly different polylogarithmic terms in the bounds in Theorem 6.3 and Lemma 6.5. Again we will define a set $\tilde{V}$ that contains enough vertices to form the desired set B by applying Theorem 5.21 to a graph $\tilde{G}$ with $V(\tilde{G}) = \tilde{V}$ and a suitable tree decomposition $(\tilde{T}, \tilde{\mathcal{X}})$. In Section 6.2, the forest induced by the set $\tilde{V}$ was not connected and, hence, it was easy to take care of some rounding effects related to its relative diameter. Now, when working with tree decompositions, the graph induced by $\tilde{V}$ might be connected and also it requires more work to reorganize the tree decomposition. More precisely, we will artificially disconnect the graph $G[\tilde{V}]$, and then glue two tree decompositions of subgraphs of $G[\tilde{V}]$ together to obtain one tree decomposition of $\tilde{G}$. Despite these difficulties, that require more work than in the tree case, there is also a simplification, namely that Case 2 and Case 3 will be more similar than in the proof of Lemma 6.2 and Case 3 does not need to be split into two subcases.

First, it is shown that a cut with the desired properties exists and then it is discussed how to compute such a cut. Let $G = (V, E)$ be an arbitrary graph on n vertices and fix some integer $m \in [n-1]$. Consider an arbitrary tree decomposition $(T, \mathcal{X})$ of G of width $t-1$ with $T = (V_T, E_T)$ and $\mathcal{X} = (X^i)_{i \in V_T}$. Without loss of generality, we may assume that $(T, \mathcal{X})$ is nonredundant, because otherwise one can contract edges of T to obtain a nonredundant tree decomposition by Proposition 5.23. Now, fix a heaviest path P

Proof of Lemma 6.2	Proof of Lemma 6.5	
forest $G = (V, E)$ on n vertices,	graph $G = (V, E)$ on n vertices,	tree decomposition $(T, \mathcal{X})$ of G of width at most $t - 1$, $T = (V_T, E_T)$, $\mathcal{X} = (X^i)_{i \in V_T}$,
assume that G is a tree,		assume that $(T, \mathcal{X})$ is nonredundant,
aim: m-cut (B, W) in G with $\operatorname{diam}^*(G[W]) \geq \operatorname{diam}^*(G)$,	aim: m-cut (B, W) in G such that $r(T', \mathcal{X}') \geq r(T, \mathcal{X})$ holds for the tree decomposition $(T', \mathcal{X}')$ induced by $G[W]$ in $(T, \mathcal{X})$,	
$P = (V_P, E_P)$ longest path in G, ends x_0 and y_0, $d = \operatorname{diam}^*(G)$, $\lvert V_P \rvert = dn$,	$R = \bigcup_{i \in V_P} X^i$, $S = V \setminus R$, $\lvert R \rvert = rn$,	$P = (V_P, E_P)$ heaviest path in T, ends i_0 and j_0, $r = w^*_{\mathcal{X}}(P)$,
T_v = components of $G - E_P$, $v \in V_P$,	$R_i = \{x \in X^i \colon$ i is the path-node of $x\}$,	T_i = components of $T - E_P$, $i \in V_P$,
$T'_v = V(T_v) \setminus \{v\}$,	$S_i = \bigcup_{j \in V(T_i)} X^j \setminus R$,	
label the vertices of G,	label the vertices of G,	no labeling of the nodes in T,
P-distance $d_P(x, y) = \lvert\{v \in V_P \colon$ v is between x and y, $v \neq y\}\rvert$.	R-distance $d_R(x, y) = \lvert\{v \in V_P \colon$ v is between x and y, $v \neq y\}\rvert$.	

Table 6.1: Overview on the notation used in the existence parts of the proofs of Lemma 6.2 and Lemma 6.5.

in $(T, \mathcal{X})$, let i_0 and j_0 be the ends of P, and note that the end i_0 of P is nonredundant as $(T, \mathcal{X})$ is nonredundant. For $i \in V_P$, define the trees T_i, the sets R, R_i, and S_i as above. Consider a P-labeling of the vertices of G, where R_{i_0} contains the vertex that received label 1 and R_{j_0} contains the vertex that received label n, and identify each vertex with its label. Furthermore, let r be the relative weight of P, which satisfies $\lvert R \rvert = rn$.

Analog to the P'-distance for a path P' in a tree T' in Section 6.2, for two vertices $x, y \in V$, define the *R-distance* of x and y as

$$d_R(x, y) = \lvert\{v \in R \setminus \{y\} \colon \text{ } v \text{ is between } x \text{ and } y\}\rvert .$$

It is easy to see that, for all $x, y \in V$ with $x \neq y$,

$$d_R(x, y) - d_R(x + 1, y) \;\in\; \{0, 1\}$$

and

$$d_R(x + 1, y) - d_R(x + 1, y + 1) \;\in\; \{0, -1\}.$$

Therefore

$$\lvert d_R(x, y) - d_R(x + 1, y + 1)\rvert \;\leq\; 1 \qquad \text{for all } x, y \in V. \tag{6.3}$$

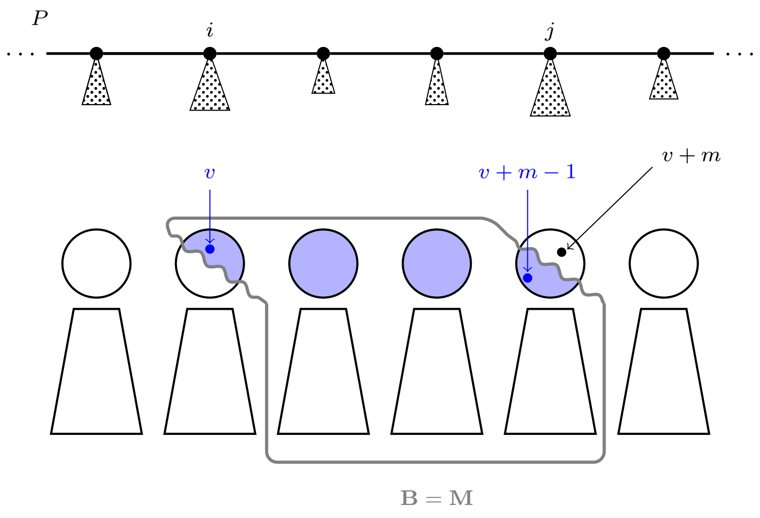


Figure 6.6: Proof of Lemma 6.5. Construction of the black set in Case 1, where $v \in R$ and $v + m \in R$. The vertex $v + m - 1$ is not necessarily in R_j.

Note that, for every $x \in V$ and all integers $\ell, \ell' \in [n-1]$, we have $d_R(x,x) = 0$ as well as

$$d_R(x, x+\ell) + d_R(x+\ell, (x+\ell)+\ell') \;=\; \begin{cases} d_R(x, x+\ell+\ell') & \text{if } \ell+\ell' < n, \\ |R| + d_R(x, x+\ell+\ell') & \text{if } \ell+\ell' \geq n. \end{cases}$$

Fix some arbitrary $x_0 \in V$ and define $x_\ell = x_0 + \ell m$ for all $\ell \in [n]$. Note that $x_n = x_0 + nm = x_0$ and

$$\sum_{\ell=1}^{n} d_R(x_{\ell-1}, x_\ell) \;=\; m|R| \;=\; mrn.$$

Therefore, there are two vertices x', $x'' \in V$ with

$$d_R(x', x'+m) \leq rm \qquad \text{and} \qquad d_R(x'', x''+m) \geq rm.$$

Using that $d_R(x,y)$ is an integer for all x, $y \in V$ to strengthen the first inequality and loosening the second one, yields

$$d_R(x', x'+m) \leq \lfloor rm \rfloor \qquad \text{and} \qquad d_R(x'', x''+m) \geq \lfloor rm \rfloor .$$

Now, (6.3) implies that there is a vertex x^* with $d_R(x^*, x^*+m) = \lfloor rm \rfloor$. If x^* or x^*+m is in R, then let $v = x^*$. Otherwise, let i and j be the path-nodes of x^* and x^*+m, respectively, and let x_R and x'_R be the smallest vertex in R_i and R_j, respectively. Note that such vertices exist, as $R_h \neq \emptyset$ for all h in P due to Proposition 5.25a). Let $s = \min\{x_R - x^*, x'_R - (x^*+m)\}$ and define $v = x^* + s$. Then, the path-nodes of v and $v+m$ are the path-nodes of x^* and x^*+m, i. e., i and j. Moreover, we have $d_R(v, v+m) = d_R(x^*, x^*+m) = \lfloor rm \rfloor$ and $v \in R$ or $v+m \in R$. Define

$$M := \{u \in V\colon\ u \text{ is between } v \text{ and } v+m-1\},$$

and note that $|M| = m$.

As in the figures in Section 5.3, the tree T is drawn in the top and the vertex sets containing vertices of the graph G are drawn underneath the corresponding node of P. More precisely, for each $i \in V_P$, the

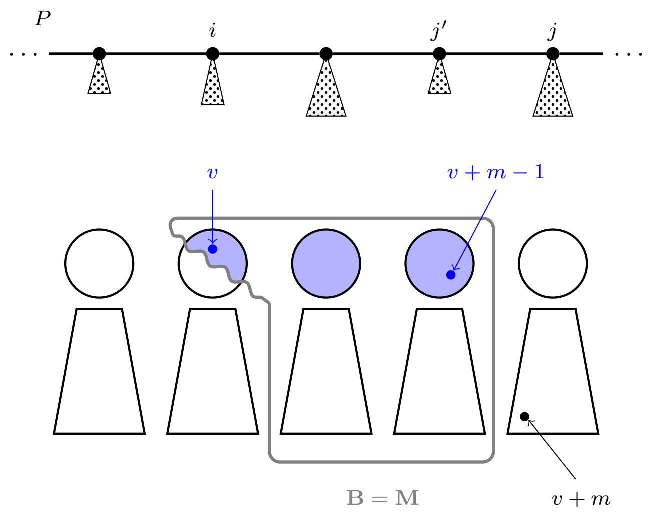


Figure 6.7: Proof of Lemma 6.5. Construction of the black set in Case 2a, where $v \in R$ and $v + m - 1 \in R$.

node i is drawn in black, the tree T_i is indicated by a triangle and the sets R_i and S_i are represented by a cycle and a trapezoid, respectively. Areas that are colored blue inside a set R_i symbolize that some vertices of R_i are counted by $d_R(v, v + m)$.

Case 1: $v \in R$ and $v + m \in R$.
Define $B := M$, $W := V \setminus B$, and note that $|B| = m$. Applying Proposition 5.26 to i and j shows that every edge that is cut by (B, W) is in $E_G(i) \cup E_G(j)$, see Figure 6.6. Therefore,

$$e_G(B, W) \;\le\; e_G(i) + e_G(j) \;\le\; 2t\Delta(G) \;\le\; \frac{t\Delta(G)}{2}\left(\left(\log_2\left(\frac{1}{r}\right)\right)^2 + 11\log_2\left(\frac{1}{r}\right) + 24\right),$$

as $r \le 1$. Let $(T', \mathcal{X}')$ be the tree decomposition induced by $G[W]$ in $(T, \mathcal{X})$. Then,

$$r(T', \mathcal{X}') \;\ge\; \frac{w_{\mathcal{X}'}(P)}{|W|} \;=\; \frac{|R \cap W|}{|W|} \;=\; \frac{|R| - d_R(v, v+m)}{|W|} \;\ge\; \frac{rn - rm}{n - m} \;=\; r,$$

as desired.

Case 2: $v \in R$ and $v + m \notin R$.
Similarly to Case 2 in the proof of Lemma 6.2, the cut $(M, V \setminus M)$ might cut too many edges in $G[S_j]$. Again, we define a set $\tilde{V}$ such that $(\tilde{V}, V \setminus \tilde{V})$ cuts few edges and show that an application of Theorem 5.21 to a graph $\tilde{G} \subseteq G[\tilde{V}]$ and the tree decomposition that $\tilde{G}$ induces in $(T, \mathcal{X})$ does not cut too many edges. Also the case when $v + m - 1 \in R$ is treated separately again for technical reasons.

Case 2a: $v + m - 1 \in R$.
As in Case 1, the cut (B, W) with $B := M$ and $W := V \setminus M$ satisfies all requirements. Denote by j' the path-node of $v + m - 1$. To prove that $e_G(B, W) \le 2t\Delta(G)$, apply Proposition 5.26 to i and j', see Figure 6.7.

Case 2b: $v + m - 1 \notin R$.
Let j' be the node before j on P. As P is nonredundant, $R_{j'}$ cannot be empty due to Proposition 5.25a). Hence, the vertex $v + m - 1$ must be in the set S_j and $S_j \cap M \neq \emptyset$. Let $\tilde{m} = 2|S_j \cap M|$ and note that $2 \le \tilde{m} \le 2m$. If $\tilde{m} \ge |S_j|$, define $B_j := S_j$ and $W_j := \emptyset$, which satisfies $\frac{1}{2}\tilde{m} \le |B_j| \le \tilde{m}$. Otherwise, using the tree decomposition that $G[S_j]$ induces in $(T, \mathcal{X})$ and the size-parameter $\tilde{m}$, Lemma 4.6 says that there is a simple approximate $\tilde{m}$-cut (B_j, W_j) in $G[S_j]$ with $e_{G[S_j]}(B_j, W_j) \le t\Delta(G)$, i. e., $B_j \dot\cup W_j = S_j$

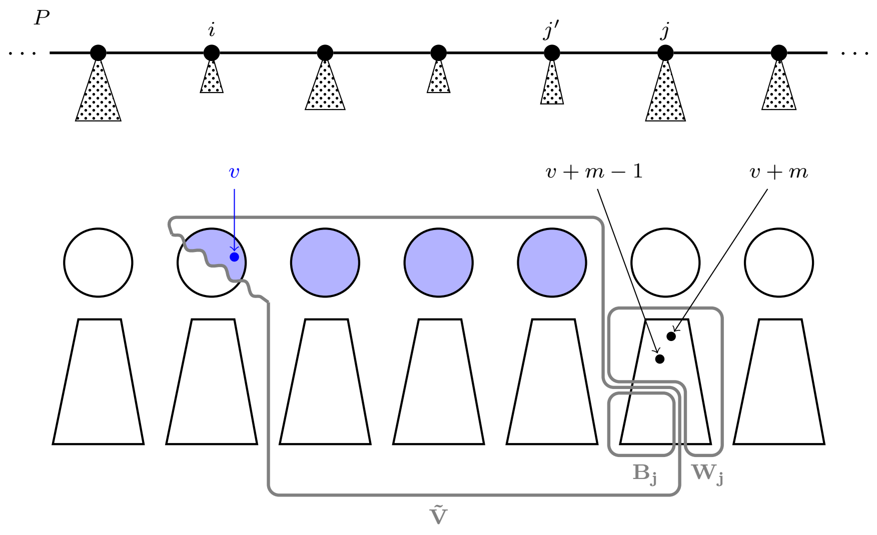


Figure 6.8: Proof of Lemma 6.5. Construction of $\tilde{V}$ in Case 2b, where $v \in R$ and $v+m \notin R$.

and $\frac{1}{2}\tilde{m} \leq |B_j| \leq \tilde{m}$. Now, define $\tilde{V} = (M \setminus S_j) \cup B_j$ and note that $\tilde{V}$ satisfies $|\tilde{V}| = m - \frac{1}{2}\tilde{m} + |B_j|$ as $M \setminus S_j$ and $B_j \subseteq S_j$ are disjoint. Using the bounds on $|B_j|$ and $\tilde{m}$ from above gives

$$m \;\leq\; |\tilde{V}| \;\leq\; m + \tfrac{1}{2}\tilde{m} \;\leq\; 2m. \tag{6.4}$$

Note that $\tilde{V}$ might contain vertices from X^j, the cluster of node j and, hence, $G[\tilde{V}]$ might be connected. Consider the graph $\tilde{G}$ obtained from $G[\tilde{V}]$ by removing all edges that are incident to some vertex in X^j. Now, $\tilde{G}$ does not contain any edge between the vertices in $B_j \subseteq S_j$ and the vertices in $\tilde{V} \setminus B_j \subseteq V \setminus S_j$ by Proposition 5.26, see Figure 6.8. Let $(\tilde{T}_1, \tilde{\mathcal{X}}_1)$ be the restriction of $(T, \mathcal{X})$ to $\tilde{T}_1 = T - (V(T_j) \setminus \{j\})$ and $\tilde{G}[\tilde{V} \setminus B_j]$, which is a tree decomposition of $\tilde{G}[\tilde{V} \setminus B_j]$ as $\tilde{V} \setminus B_j$ contains no vertex that is in the set $S_j = \bigcup_{h \in T_j \setminus \{j\}} X^h \setminus R$. Observe that $w_{\tilde{\mathcal{X}}_1}(\tilde{P}_1) \geq d_R(v, v+m)$ holds for $\tilde{P}_1 := P$, as every vertex counted by $d_R(v, v+m)$ is in $\tilde{V} \setminus B_j = M \setminus S_j$. Furthermore, let $(\tilde{T}_2, \tilde{\mathcal{X}}_2)$ be the restriction of $(T, \mathcal{X})$ to $\tilde{T}_2 = T_j$ and $G[B_j]$, which is a tree decomposition of $G[B_j]$, since every vertex in $B_j \subseteq S_j$ is only in clusters X^h with $h \in T_j \setminus \{j\}$ due to Property (T3') of tree decompositions. Let h_0 be a node in $\tilde{T}_2$ whose cluster in $\tilde{\mathcal{X}}_2$ is non-empty. Such a node h_0 exists as $B_j \neq \emptyset$. Denote by $\tilde{P}_2$ the path consisting only of the node h_0 and observe that $w_{\tilde{\mathcal{X}}_2}(\tilde{P}_2) \geq 1$. Now, let $\tilde{T}$ be the tree obtained from taking one copy of the tree $\tilde{T}_1$ and one copy of the tree $\tilde{T}_2$ with disjoint node sets and adding an edge between j_0 and h_0. Denote by $\tilde{\mathcal{X}}$ the union of the cluster collections $\tilde{\mathcal{X}}_1$ and $\tilde{\mathcal{X}}_2$, with an adjustment of the indices, and note that $(\tilde{T}, \tilde{\mathcal{X}})$ is a tree decomposition of $\tilde{G}$. Furthermore, let $\tilde{P}$ be the path corresponding to the nodes of $\tilde{P}_1$ and $\tilde{P}_2$ after the adjustment of the indices. Then,

$$w_{\tilde{\mathcal{X}}}(\tilde{P}) \;\geq\; w_{\tilde{\mathcal{X}}_1}(\tilde{P}_1) + w_{\tilde{\mathcal{X}}_2}(\tilde{P}_2) \;\geq\; d_R(v, v+m) + 1 \;\geq\; rm,$$

because the clusters in $\tilde{\mathcal{X}}_1$ and $\tilde{\mathcal{X}}_2$ are pairwise disjoint. Now, (6.4) implies that

$$r(\tilde{T}, \tilde{\mathcal{X}}) \;\geq\; \frac{w_{\tilde{\mathcal{X}}}(\tilde{P})}{|\tilde{V}|} \;\geq\; \frac{rm}{2m} \;\geq\; \tfrac{1}{2}r.$$

Using the tree decomposition $(\tilde{T}, \tilde{\mathcal{X}})$ and size-parameter m, Theorem 5.21 implies that there is an m-cut $(\tilde{B}, \tilde{W})$ in $\tilde{G}$ with

$$e_{\tilde{G}}(\tilde{B}, \tilde{W}) \;\leq\; \frac{t\Delta(\tilde{G})}{2} \left(\left(\log_2 \left(\frac{2}{r} \right) \right)^2 + 9 \log_2 \left(\frac{2}{r} \right) + 8 \right) \tag{6.5}$$

$$\leq \frac{t\Delta(G)}{2}\left(\left(\log_2\left(\frac{1}{r}\right)\right)^2 + 11\log_2\left(\frac{1}{r}\right) + 18\right).$$

Now, define $B := \tilde{B}$ and $W := V \setminus \tilde{B}$. Furthermore, denote by $(T', \mathcal{X}')$ the tree decomposition induced by $G[W]$ in $(T, \mathcal{X})$. By construction, there are exactly $d_R(v, v+m)$ vertices of R in $\tilde{V}$ and, hence, at most $d_R(v, v+m)$ vertices of R are in $B \subseteq \tilde{V}$. Consequently, at least $|R| - d_R(v, v+m)$ vertices of R are in W and

$$w_{\mathcal{X}'}(P) \;\geq\; |R \cap W| \;\geq\; |R| - d_R(v, v+m) \;\geq\; r(n-m),$$

as well as

$$r(T', \mathcal{X}') \;\geq\; \frac{w_{\mathcal{X}'}(P)}{|W|} \;\geq\; \frac{r(n-m)}{n-m} \geq\; r.$$

Next, the width of the cut (B, W) in G is analyzed. Let $\hat{G} := G - E_G(i) - E_G(j)$. The graph $\hat{G}$ contains no edges between the sets $M \setminus S_j$, S_j, and $V \setminus (M \cup S_j)$ by Proposition 5.26 applied to i and j. So, $e_G(\tilde{V}, V \setminus \tilde{V}) \leq 2t\Delta(G) + e_{\hat{G}}(\tilde{V}, V \setminus \tilde{V})$. As mentioned above, the cut (B_j, W_j) cuts at most $t\Delta(G)$ edges in $G[S_j]$. Using that $\tilde{V} = (M \setminus S_j) \cup B_j$ yields

$$e_G(\tilde{V}, V \setminus \tilde{V}) \;\leq\; 2t\Delta(G) + e_{\hat{G}}(\tilde{V}, V \setminus \tilde{V}) \;\leq\; 2t\Delta(G) + e_{G[S_j]}(B_j, W_j) \;\leq\; 3t\Delta(G).$$

Note that the previous estimation also counts all edges that were removed from $G[\tilde{V}]$ when constructing $\tilde{G}$, as these edges are in $E_G(j)$. Using (6.5) to count the edges cut by $(\tilde{B}, \tilde{W})$ in $\tilde{G}$ yields

$$\begin{aligned} e_G(B, W) \;&\leq\; 3t\Delta(G) + \frac{t\Delta(G)}{2}\left(\left(\log_2\left(\frac{1}{r}\right)\right)^2 + 11\log_2\left(\frac{1}{r}\right) + 18\right) \\ &\leq\; \frac{t\Delta(G)}{2}\left(\left(\log_2\left(\frac{1}{r}\right)\right)^2 + 11\log_2\left(\frac{1}{r}\right) + 24\right), \end{aligned}$$

as desired.

Case 3: $v \notin R$ and $v + m \in R$.
This case is similar to Case 2b, but not completely analogous, as in the proof of Lemma 6.2. Instead of splitting $S_j = B_j \cup W_j$, now S_i is split. To do so, let $\tilde{m} := 2|S_i \cap M|$, which satisfies $2 \leq \tilde{m} \leq 2m$ as $v \in S_i \cap M$. As in Case 2b, let (B_i, W_i) be a cut in $G[S_i]$ with $B_i \cup W_i = S_i$, $e_{G[S_i]}(B_i, W_i) \leq t\Delta(G)$, and $\frac{1}{2}\tilde{m} \leq |B_i| \leq \tilde{m}$. Then, $\tilde{V} := (M \setminus S_i) \cup B_i$ satisfies $m \leq |\tilde{V}| \leq 2m$. Consider the graph $\tilde{G}$ obtained from $G[\tilde{V}]$ by removing all edges that are incident to some vertex in X^i and note that $\tilde{G}$ does not contain any edge between the vertices in B_i and the vertices in $\tilde{V} \setminus B_i$, see Figure 6.9. Similarly to Case 2b, a tree decomposition $(\tilde{T}, \tilde{\mathcal{X}})$ of $\tilde{G}$ with $r(\tilde{T}, \tilde{\mathcal{X}}) \geq \frac{1}{2}r$ can be constructed from a tree decomposition of $\tilde{G}[\tilde{V} \setminus B_i]$ and a tree decomposition of $\tilde{G}[B_i]$ by using that $d_R(v, v+m)$ vertices of R are in $\tilde{V} \setminus B_i$ and $B_i \neq \emptyset$. Using the tree decomposition $(\tilde{T}, \tilde{\mathcal{X}})$ and the size-parameter m, Theorem 5.21 implies that there is an m-cut $(\tilde{B}, \tilde{W})$ in $\tilde{G}$ with

$$e_{\tilde{G}}(\tilde{B}, \tilde{W}) \;\leq\; \frac{t\Delta(G)}{2}\left(\left(\log_2\left(\frac{1}{r}\right)\right)^2 + 11\log_2\left(\frac{1}{r}\right) + 18\right).$$

Now, define $B := \tilde{B}$ and $W := V \setminus \tilde{B}$, which satisfies $|B| = m$. Using that the set $\tilde{V}$ contains $d_R(v, v+m)$ vertices of R, one can show analogously to Case 2b that the tree decomposition $(T', \mathcal{X}')$ induced by $G[W]$ in $(T, \mathcal{X})$ satisfies $r(T', \mathcal{X}') \geq r$. Analog to Case 2b, one can show that the width of the cut (B, W) in G is within the desired bound, by considering the graph $\hat{G} := G - E_G(i) - E_G(j)$, which does not contain any edges between S_i, $M \setminus S_i$, and $V \setminus (M \cup S_i)$.

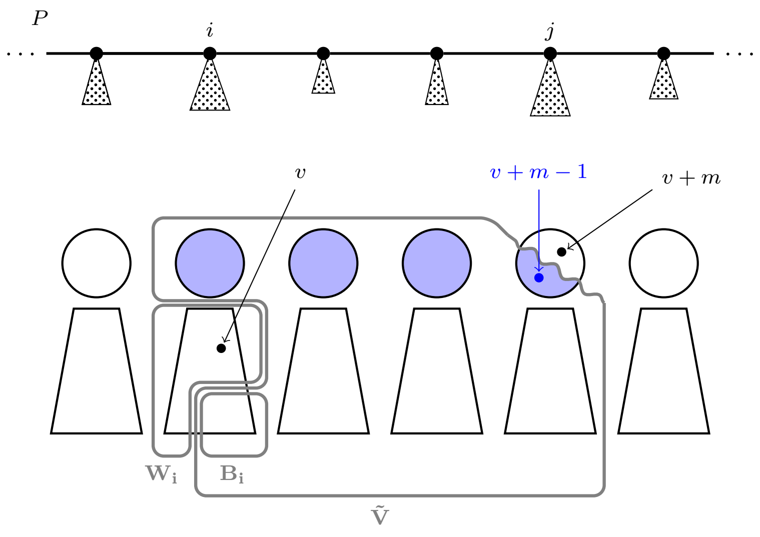


Figure 6.9: Proof of Lemma 6.5. Construction of $\tilde{V}$ in Case 3, where $v \notin R$ and $v+m \in R$. Note that $v+m-1$ is not necessarily in R.

Next, it is discussed how to implement an algorithm that computes such a cut (B, W) in a graph G. Recall that the algorithm in Lemma 6.5 neither receives G nor $(T, \mathcal{X})$ as input, but receives a tree decomposition $(T_0, \mathcal{X}_0)$ of some graph G_0 on n_0 vertices with $V(G_0) = [n_0]$ and $G \subseteq G_0$ as well as a list L_G of the vertices of G as input. First, the algorithm computes a tree decomposition of G, more precisely, the tree decomposition $(T, \mathcal{X})$ that is induced by G in $(T_0, \mathcal{X}_0)$ and then follows the construction presented above, which is summarized in Algorithm 6.2. As in the implementation of the algorithm contained in Theorem 5.21, Algorithm 6.2 does not identify vertices with their labels. From now on, $L(x)$ is used to refer to the label of a vertex $x \in V$ and $L^{-1}(a)$ is used to refer to the vertex which received label $a \in [n]$. Furthermore, the definition of the R-distance is adjusted such that it refers to two vertex names, i. e.,

$$d_R(x, y) = |\{v \in R\colon\ L(v) \text{ is between } L(x) \text{ and } L(y),\ v \neq y\}|$$

for all $x, y \in V(G)$. As usually, only the set B of the desired cut (B, W) is returned and throughout the algorithm, all sets are stored as unordered lists of vertices.

The aim is to show that Algorithm 6.2 can be implemented to run in $\mathcal{O}(\|(T_0, \mathcal{X}_0)\|)$ time. To do so, it suffices to show that every line can be executed in time proportional to m, n, n_0, $\|(T, \mathcal{X})\|$, or $\|(T_0, \mathcal{X}_0)\|$ as $\|(T, \mathcal{X})\| \leq \|(T_0, \mathcal{X}_0)\|$ and $m \leq n \leq n_0 \leq \|(T_0, \mathcal{X}_0)\|$. Before executing Line 1, the algorithm sets up a bijection between the vertices in G and $[n]$, which takes $\mathcal{O}(n_0)$ time by Lemma 2.21a). Then, $(T, \mathcal{X})$ is computed, which takes time proportional to $(T_0, \mathcal{X}_0)$ according to Proposition 2.31b) and Lemma 2.21c), and while doing so the algorithm renames the vertices with the bijection such that, from now on, we may assume that $V(G) = [n]$. Then, Line 2 and Line 3 each take $\mathcal{O}(\|(T, \mathcal{X})\|)$ time by Proposition 2.32b), Lemma 5.32, and Lemma 5.35. Denote by A_L, A_V, A_R, A_P, and L_P the arrays and the list of the set of P-parameters. Recall that all array indices refer to vertex names, except the ones in A_V. Similarly to the implementation of Algorithm 6.1 in the proof of Lemma 6.2, for each x in V, the number $d_1(x)$ as defined in Line 4 is the number of vertices in R, whose label is strictly smaller than $L(x)$. So the algorithm first sets $d_1(x) = 0$ for the vertex $x = L^{-1}(1)$. Then it goes through all other vertices x in increasing order of their labels, i. e., in the order in which they are stored in A_V, and sets $d_1(x) = d_1(x^-) + 1$

Algorithm 6.2: Computing an m-cut with the properties in Lemma 6.5.

```
   Input: a list of the vertices of a graph $G$ on $n$ vertices, a tree decomposition $(T_0, \mathcal{X}_0)$ of a graph $G_0$
          with $G \subseteq G_0$ and $V(G_0) = [n_0]$, and an integer $m \in [n-1]$.
   Output: an $m$-cut $(B, W)$ in $G$ with the properties in Lemma 6.5.
1  Compute the tree decomposition $(T, \mathcal{X})$ induced by $G$ in $(T_0, \mathcal{X}_0)$;
2  Turn $(T, \mathcal{X})$ into a nonredundant tree decomposition;
3  Compute a heaviest path $P$ in $T$ with respect to $\mathcal{X}$, its relative weight $r = r(T, \mathcal{X})$, and the set of
     $P$-parameters of $G$. To state the algorithm, let $R$ be the set of root vertices of $P$, denote by $L(x)$ the
     label of vertex $x \in V$ and by $L^{-1}(a)$ the vertex that received label $a \in [n]$;
4  For each $x \in V(G)$ compute $d_1(x) = d_R(L^{-1}(1), x)$;
5  Determine a vertex $v \in V$ with $d_R(v, v+m) = \lfloor rm \rfloor$ and $v \in R$ or $L^{-1}(L(v)+m) \in R$;
6  $M \leftarrow \{w \in V(G)\colon\ L(w)$ is between $L(v)$ and $L(v)+m-1\}$;
7  If $v \in R$ then
8      If $L^{-1}(L(v)+m) \in R$ or $L^{-1}(L(v)+m-1) \in R$ then
9          $B \leftarrow M$;
10         Return $(B, V(G) \setminus B)$;
11     Endif
12     Let $h$ be the path-vertex of $L^{-1}(L(v)+m)$;
13 Else
14     Let $h$ be the path-vertex of $v$;
15 Endif
16 $\tilde{m} \leftarrow 2|S_h \cap M|$;
17 Let $(B_h, W_h)$ be a cut in $G[S_h]$ with $e_{G[S_h]}(B_h, W_h) \leq t\Delta(G)$ and $\frac{1}{2}\tilde{m} \leq |B_h| \leq \tilde{m}$;
18 $\tilde{V} \leftarrow (M \setminus S_h) \cup B_h$;
19 Let $(\tilde{T}, \tilde{\mathcal{X}})$ be a tree decomposition of $\tilde{G} = G[\tilde{V}] - E_G(h)$ with $r(\tilde{T}, \tilde{\mathcal{X}}) \geq \frac{1}{2}r$;
20 Compute an $m$-cut $(\tilde{B}, \tilde{W})$ in $\tilde{G}$ as in Theorem 5.39 by using the tree decomposition $(\tilde{T}, \tilde{\mathcal{X}})$;
21 $B \leftarrow \tilde{B}$;
22 Return $(B, V(G) \setminus B)$;
```

if $x^- \in R$ and $d_1(x) = d_1(x^-)$ otherwise, where x^- denotes the vertex with label $L(x) - 1$. Hence, Line 4 takes time proportional to n, as the algorithm can check in constant time whether a vertex is in R by using the array A_R. Using $d_1(.)$, it is easy to compute $d_R(x, x+m)$ and $d_R(x, x-m)$ in constant time for an arbitrary $x \in R$ as $d_R(x, y) = d_1(y) - d_1(x)$ for all $x, y \in V$ with $L(x) < L(y)$. To determine a vertex v with the property in Line 5, the algorithm goes through all vertices in R, i. e., it traverses the array A_R and considers only vertices x with $A_R(x) = 1$, and checks whether $d_R(x, x-m) = \lfloor rm \rfloor$ or $d_R(x, x+m) = \lfloor rm \rfloor$ is satisfied. As argued above, a vertex v with the property in Line 5 exists always. Hence, Line 5 can be executed in time proportional to n_0. The set M in Line 6 can be read off the array A_V in $\mathcal{O}(m)$ time. The conditions in Line 7 and Line 8 can be checked in constant time by using the arrays A_R, A_L, and A_V. Line 9 and Line 10, if executed, take time proportional to m. Note that here, the algorithm needs to convert the current vertex names of G back to the vertex names in G_0, which takes constant time for each vertex by Lemma 2.21b).

From now on, assume that Lines 9-10 are not executed. So either Line 12 or Line 14 is executed and each of them takes constant time by using the array A_P. Furthermore, Lines 16-22 are executed and discussed now. To compute $\tilde{m}$ in Line 16, the algorithm traverses the set M and checks for each vertex $v \in M$

whether it is in S_h, which takes constant time for each $v \in M$ by Proposition 5.34b). All other vertices are in $M \setminus S_h$ and are collected in a list L^1_V, which will be useful for Line 18. All in all, Line 16 takes time proportional to m. To execute Line 17, the algorithm first determines a list of the vertices in S_h in $\mathcal{O}(n)$ time by going through all vertices x in G, i. e., going through the set $[n]$, and checking whether $x \in S_h$ as in Proposition 5.34b). While doing so, the algorithm also determines $n_h := |S_h|$. If $|S_h| \leq \tilde{m}$, it sets $B_h = S_h$. Otherwise, (B_h, W_h) is a simple approximate $\tilde{m}$-cut in $G[S_h]$ and can be computed with the algorithm contained in Lemma 4.6. To do so, a tree decomposition of $G[S_h]$ needs to be provided and the vertex set of $G[S_h]$ needs to be $[n_h]$. Both can be achieved in $\mathcal{O}(n + \|(T, \mathcal{X})\|)$ time by Lemma 2.21 and Proposition 2.31b). Hence, Line 17 takes $\mathcal{O}(n + \|(T, \mathcal{X})\|) = \mathcal{O}(\|(T, \mathcal{X})\|)$ time, including the time needed to convert back the names of the vertices in the set B_h. Then, Line 18 is a concatenation of disjoint lists and takes constant time.

To implement Line 19, recall that $(\tilde{T}, \tilde{\mathcal{X}})$ was constructed from two tree decompositions, as the tree decomposition induced by $\tilde{G}$ in $(T, \mathcal{X})$ might not satisfy the requirements. So the algorithm computes the sizes n_1 and n_2 of the sets $M \setminus S_h$ and B_h, respectively, as well as a bijection from $M \setminus S_h$ to $[n_1]$ and a bijection from B_h to $[n_1 + n_2] \setminus [n_1]$, which takes $\mathcal{O}(n_1 + n_2) = \mathcal{O}(n)$ time together by Lemma 2.21a). Note that both bijections together result in a bijection renaming the vertices of $\tilde{G}$ to $[n_1 + n_2]$. Then, the algorithm computes the restriction $(\tilde{T}_1, \tilde{\mathcal{X}}_1)$ of $(T, \mathcal{X})$ to $\tilde{T}_1 := T - (V(T_h) \setminus \{h\})$ and $G[M \setminus S_h]$, and the restriction $(\tilde{T}_2, \tilde{\mathcal{X}}_2)$ of $(T, \mathcal{X})$ to T_h and $G[S_h]$. Using that the tree T_h is known from the set of P-parameters, it follows that this takes $\mathcal{O}(\|(T, \mathcal{X})\|)$ time by Lemma 2.21 and Proposition 2.31b). To ensure that the node sets of $\tilde{T}_1$ and $\tilde{T}_2$ are disjoint, the algorithm renames each of them, which takes time proportional to $|V(\tilde{T}_1)| + |V(\tilde{T}_2)| \leq \|(\tilde{T}, \tilde{\mathcal{X}})\|$ by Lemma 2.22. Then, the algorithm determines a node h_0 in $\tilde{T}_2$ whose cluster in $\tilde{\mathcal{X}}_2$ is non-empty, and joins $\tilde{T}_1$ and $\tilde{T}_2$ by inserting the edge $\{j_0, h_0\}$, which together takes at most $\mathcal{O}(\|(\tilde{T}_2, \tilde{\mathcal{X}}_2)\|)$ time. As described in the construction, the new tree $\tilde{T}$ obtained from $\tilde{T}_1$ and $\tilde{T}_2$ together with the union of the clusters $\tilde{\mathcal{X}}_1$ and $\tilde{\mathcal{X}}_2$ form a tree decomposition $(\tilde{T}, \tilde{\mathcal{X}})$ of $\tilde{G}$ with the desired property for Line 19. Then, an application of the algorithm in Theorem 5.39 to $\tilde{G}$ with the tree decomposition $(\tilde{T}, \tilde{\mathcal{X}})$ and the size-parameter m takes time proportional to $\|(\tilde{T}, \tilde{\mathcal{X}})\| = \|(\tilde{T}_1, \tilde{\mathcal{X}}_1)\| + \|(\tilde{T}_2, \tilde{\mathcal{X}}_2)\|$. So all in all, Line 20 takes $\mathcal{O}(\|(T, \mathcal{X})\| + \|(\tilde{T}_1, \tilde{\mathcal{X}}_1)\| + \|(\tilde{T}_2, \tilde{\mathcal{X}}_2)\|) = \mathcal{O}(\|(T, \mathcal{X})\|)$ time, including to convert back the vertex names in $\tilde{B}$. Finally to return the set $B := \tilde{B}$, the algorithm needs to convert back the vertex names, which takes constant time for each vertex by Lemma 2.21b).

Chapter 7

Open Problems

Finally, this last chapter presents a selection of open problems. Probably the most intriguing open questions concerning the Minimum Bisection Problem are the following two. Is it NP-hard to approximate the width of a minimum bisection within a constant factor? Does the Minimum Bisection Problem remain NP-hard when restricted to planar graphs? The following four open questions are closely related to the methods used in this thesis.

Question 7.1.
Consider a class $\mathcal{G}$ of bounded-degree graphs with strongly sublinear separators, which means that there are constants $\Delta_0 \in \mathbb{N}$, $\sigma > 0$, and $\delta < 1$ such that every graph G in $\mathcal{G}$ on n vertices satisfies $\Delta(G) \leq \Delta_0$ and has a $\frac{1}{2}$-separator of size at most σn^δ. Furthermore, assume that $\mathcal{G}$ is closed under taking subgraphs. The techniques applied in Section 3.1 can be used to show that there is a constant c_σ such that every graph $G \in \mathcal{G}$ on n vertices satisfies $\mathrm{MinBis}(G) \leq c_\sigma n^\delta$. Furthermore, similar to Theorem 3.9 and Theorem 3.13, it seems that, for every $c > 0$, there is a $\gamma = \gamma(c, \Delta_0) > 0$ such that, for every graph $G \in \mathcal{G}$ on n vertices, the following holds

$$\mathrm{MinBis}(G) \geq cn^\delta \qquad \Rightarrow \qquad \mathrm{tw}(G) \geq \gamma n^\delta - 1.$$

What upper bounds on the tree-width of the graphs in the class $\mathcal{G}$ can be derived? Does large minimum bisection width imply large tree-width for graphs in $\mathcal{G}$? Recall that, for bounded-degree planar graphs, large minimum bisection width implies large tree-width, which is equivalent to containing a large grid as a minor. Is there some property for $\mathcal{G}$ that is equivalent to having large tree-width?

Question 7.2.
As discussed in the beginning of Section 3.3, the algorithm in Theorem 1.10 is a constant-factor approximation for the Minimum Bisection Problem in bounded-degree planar graphs that are (γ, k, ℓ)-grid-homogeneous for certain parameters γ, k, and ℓ. The property of being grid-homogeneous was only used to derive a lower bound on the minimum bisection width but not when constructing a bisection. Consider a class $\mathcal{G}$ of bounded-degree planar graphs, where each graph $G \in \mathcal{G}$ is (γ, k, ℓ)-grid-homogeneous for some $\gamma < \frac{1}{2}$, some constant ℓ, and some k that depends on the number of vertices of G. What upper bounds can be derived for the minimum bisection width of the graphs in $\mathcal{G}$? Is it possible to use these ideas algorithmically, i. e., when given a graph G and a certificate that G is (γ, k, ℓ)-grid-homogeneous, is it possible to compute a bisection whose width is within such a bound in polynomial time?

Question 7.3.
It is open whether, for planar graphs G, it is NP-hard to determine $\mathrm{grid}(G)$, which is defined as the largest integer k such that G contains a $k \times k$ grid as a minor. Can the ideas used in Section 3.3.4 to show that the HGM Problem is NP-hard be modified to show that, for planar graphs G, it is NP-hard to determine $\mathrm{grid}(G)$? Can these ideas be used to show that other more restricted versions that ask for containing a grid as a minor with additional properties are NP-hard? For example, consider the following two problems. Given a planar graph G, determine the largest integer k such that G contains a subgraph H that is a subdivision of a $k \times k$ grid. Given a planar graph G and an integer $\ell \in \mathbb{N}$, determine the largest integer k such that G contains the $k \times k$ grid as an ℓ-shallow minor, which is defined as follows. For $\ell \in \mathbb{N}$, a graph G contains a graph H as an ℓ-shallow minor if there is a collection of nonempty pairwise disjoint sets $(M_x)_{x \in V(H)}$ with $M_x \subseteq V(G)$ for each $x \in V(H)$ such that the following holds:

- For each $x \in V(H)$, the graph $G[M_x]$ has radius at most ℓ, i. e., there is a vertex $y \in M_x$ such that $\mathrm{dist}_{G[M_x]}(y, y') \leq \ell$ for all $y' \in M_x$.
- If each set M_x is contracted to one vertex x, then the graph H is obtained.

Question 7.4.
The algorithms for computing a k-section of small width in trees and tree-like graphs, that were presented in Theorem 6.3 and Theorem 6.4, respectively, do not run in linear time. Is it possible to speed up these algorithms to run in linear time?

Appendix A

Generalizing the Concept of Path-Prosperous Graphs

In the following, the definition of path-prosperous graphs is generalized and a lower bound on the minimum bisection width in these graphs is proved. The following definition is a generalization of Definition 3.20, which relaxes property (P1). To this purpose, the collection of clusters is not anymore required to be a partition of V', but all vertices in V' appear in almost the same amount of clusters. The parameter ε captures this deviation and the parameter a denotes the average number of times a vertex $v \in V'$ appears in a cluster in $\mathcal{X}$.

Definition A.1 ($(\gamma, \varepsilon, k, c)$-path-prosperous).
Let $0 \leq \varepsilon < 1$, $0 \leq \gamma < 1$, $k \in \mathbb{N}$, $0 < c \leq 1$. A graph $G = (V, E)$ is called $(\gamma, \varepsilon, k, c)$-path-prosperous if it contains a subgraph $G' = (V', E')$ with $|V'| \geq (1-\gamma)|V|$ such that there exists a collection of clusters $\mathcal{X} = (X^i)_{i \in I}$ with $X^i \subseteq V'$ satisfying the following properties. First, let

$$I_v := \left\{ i \in I : v \in X^i \right\} \qquad \text{for all } v \in V' \qquad \text{and} \qquad a := \frac{1}{|V'|} \sum_{v \in V'} |I_v| \, .$$

(P1') for all $v \in V'$, the set I_v satisfies $(1-\varepsilon)a \leq |I_v| \leq (1+\varepsilon)a$,

(P2) for all $i \in I$, the set X^i satisfies $|X^i| \geq k$,

(P3) for all $d \in [k]$, for all $i_1, i_2 \in I$, and for all $Z_1 \subseteq X^{i_1}$, $Z_2 \subseteq X^{i_2}$ with $|Z_1| = |Z_2| = d$, there exist at least cd edge-disjoint Z_1,Z_2-paths in G'.

Observe that for $\varepsilon = 0$ and $a = 1$, that is, whenever each vertex $v \in V'$ belongs to exactly one of the sets X^i with $i \in I$, property (P1') in the above definition is equivalent to property (P1) in Definition 3.20. Consider a (γ, k, c)-path-prosperous graph G, where $\mathcal{X} = (X^i)_{i \in I}$ is a collection of clusters with the properties in Definition 3.20 and further consider a minimum bisection (B, W) in G. In the proof of Lemma 3.22, it was easy to find two indices i_B and i_W such that the corresponding clusters in $\mathcal{X}$ contained many vertices in B and W, respectively. Such indices also exist for $(\gamma, \varepsilon, k, c)$-path-prosperous graphs, as is shown in the next lemma. Its proof constitutes most of the work needed to derive a lower bound on the minimum bisection width in $(\gamma, \varepsilon, k, c)$-path-prosperous graphs. It uses the following notion: For a set S

and a subset $S' \subseteq S$, let

$$\mathbb{1}_{S'}(x) := \begin{cases} 1 & \text{if } x \in S', \\ 0 & \text{otherwise} \end{cases}$$

for all $x \in S$ and note that $\sum_{x\in S} \mathbb{1}_{S'}(x) = |S'|$ holds for all $S' \subseteq S$.

Lemma A.2.
For all $0 \leq \gamma < \frac{1}{2}$ and for all $0 \leq \varepsilon < 1-\gamma$ there is an $\alpha_{\gamma,\varepsilon} > 0$ such that the following holds. For every graph $G = (V, E)$ with $|V|$ even, every subgraph $G' = (V', E')$ with $|V'| \geq (1-\gamma)|V|$ and every collection of clusters $\mathcal{X} = (X^i)_{i\in I}$ with $X^i \subseteq V'$ that satisfies (P1') with respect to ε, the following is true.
For every bisection (B, W) of G, there are indices $i_B, i_W \in I$ with

$$\left|X^{i_B} \cap B\right| \;\geq\; \alpha_{\gamma,\varepsilon} \left|X^{i_B}\right| \qquad \textit{and} \qquad \left|X^{i_W} \cap W\right| \;\geq\; \alpha_{\gamma,\varepsilon} \left|X^{i_W}\right|.$$

In particular, one can choose

$$\alpha_{\gamma,\varepsilon} = \frac{(1-2\gamma)(1-\gamma-\varepsilon)}{(1-\gamma)(1-2\gamma)+1}.$$

Proof. Fix some $0 \leq \gamma < \frac{1}{2}$ and $0 \leq \varepsilon < 1-\gamma$. Let $G = (V, E)$ be an arbitrary graph with $|V|$ even, $G' = (V', E')$ a subgraph of G with

$$|V'| \;\geq\; (1-\gamma)\,|V|, \tag{A.1}$$

and $\mathcal{X} = (X^i)_{i\in I}$ with $X^i \subseteq V'$ a collection of clusters that satisfies (P1') with respect to ε. As in Definition A.1, for each $v \in V'$ define $I_v := \{i \in I : v \in X^i\}$ and let $a := \frac{1}{|V'|}\sum_{v\in V'} |I_v|$. Then,

$$\begin{aligned} a\,|V'| \;&=\; \sum_{v\in V'} |I_v| \;=\; \sum_{v\in V'} \sum_{i\in I} \mathbb{1}_{X^i}(v) \\ &=\; \sum_{i\in I} \sum_{v\in V'} \mathbb{1}_{X^i}(v) \;=\; \sum_{i\in I} \left|X^i\right|. \end{aligned} \tag{A.2}$$

Furthermore, (P1') guarantees that

$$(1-\varepsilon)a \;\leq\; |I_v| \;\leq\; (1+\varepsilon)a \qquad\qquad \text{for all } v \in V'. \tag{A.3}$$

Define

$$\alpha_{\gamma,\varepsilon} \;:=\; \frac{(1-2\gamma)(1-\gamma-\varepsilon)}{(1-\gamma)(1-2\gamma)+1}. \tag{A.4}$$

Let us quickly argue that $\alpha_{\gamma,\varepsilon} > 0$. The parameters γ and ε satisfy $1-\gamma-\varepsilon > 0$ and $1-2\gamma > 0$. Therefore, both the numerator and denominator in the fraction in (A.4) are positive and, hence, $\alpha_{\gamma,\varepsilon} > 0$.

Next, it is shown that $\alpha_{\gamma,\varepsilon} \leq \frac{1}{2}$. Clearly,

$$(1-\gamma)(1-2\gamma) \;\leq\; 1 \;\leq\; 1 + 2\varepsilon(1-2\gamma)$$

holds and consequently

$$\begin{aligned} & 2(1-2\gamma)(1-\gamma) - 2\varepsilon(1-2\gamma) \;\leq\; (1-\gamma)(1-2\gamma)+1 \\ \Leftrightarrow \qquad & 2(1-2\gamma)(1-\gamma-\varepsilon) \;\leq\; (1-\gamma)(1-2\gamma)+1, \end{aligned}$$

which shows that $\alpha_{\gamma,\varepsilon} \leq \frac{1}{2}$.

Let $B \mathbin{\dot\cup} W = V$ be an arbitrary bisection in G. As $|V|$ is even, $|B| = |W| = \frac{1}{2}|V|$. Fix some arbitrary index $j \in I$. Each vertex in X^j is either in $X^j \cap B$ or in $X^j \cap W$, and therefore $|X^j \cap B| \geq \frac{1}{2}|X^j|$ or $|X^j \cap W| \geq \frac{1}{2}|X^j|$ holds. Since $\alpha_{\gamma,\varepsilon} \leq \frac{1}{2}$, assume without loss of generality that $|X^j \cap B| \geq \alpha_{\gamma,\varepsilon}|X^j|$ and choose $i_B = j$.

Assume for a contradiction that there is no $i \in I$ with $|X^i \cap W| \geq \alpha_{\gamma,\varepsilon}|X^i|$, i. e.,

$$\left|X^i \cap W\right| \ < \ \alpha_{\gamma,\varepsilon}\left|X^i\right| \qquad \text{for all } i \in I. \tag{A.5}$$

Then,

$$\left|X^i \cap B\right| \ > \ (1-\alpha_{\gamma,\varepsilon})\left|X^i\right| \qquad \text{for all } i \in I, \tag{A.6}$$

as for every $i \in I$, each vertex in X^i is either in $X^i \cap B$ or in $X^i \cap W$. To derive the desired contradiction, define

$$\mathcal{S}_B := \left\{(v,i) \in (V' \cap B) \times I : v \in X^i\right\}$$
$$\mathcal{S}_W := \left\{(v,i) \in (V' \cap W) \times I : v \in X^i\right\}.$$

Since $\gamma < \frac{1}{2}$, there are strictly more than $\frac{1}{2}|V|$ vertices in V'. Hence, some vertex in V' must lie in B and some vertex in V' must lie in W. Therefore $V' \cap B \neq \emptyset$, $V' \cap W \neq \emptyset$, and so the following maximum and minimum are being taken over non-empty sets

$$a_B^{\max} := \max\left\{|I_v| : v \in V' \cap B\right\}, \qquad a_W^{\min} := \min\left\{|I_v| : v \in V' \cap W\right\}. \tag{A.7}$$

Next, by estimating the sizes of the sets $\mathcal{S}_B$ and $\mathcal{S}_W$, a lower bound for $a_B^{\max}$ and an upper bound for $a_W^{\min}$ are derived. The set $\mathcal{S}_B$ satisfies

$$|\mathcal{S}_B| \ = \sum_{v \in V' \cap B} \sum_{i \in I} \mathbb{1}_{X^i}(v) \ = \sum_{v \in V' \cap B} |I_v| \ \overset{\text{(A.7)}}{\leq} \ |V' \cap B|\, a_B^{\max}$$
$$\leq \ \tfrac{1}{2}|V|\, a_B^{\max} \ \overset{\text{(A.1)}}{\leq} \ \frac{1}{2(1-\gamma)}|V'|\, a_B^{\max}$$

and

$$|\mathcal{S}_B| \ = \sum_{i \in I} \sum_{v \in V' \cap B} \mathbb{1}_{X^i}(v) \ = \sum_{i \in I}\left|X^i \cap B\right| \ \overset{\text{(A.6)}}{>} \ \sum_{i \in I}(1-\alpha_{\gamma,\varepsilon})\left|X^i\right| \ \overset{\text{(A.2)}}{=} \ (1-\alpha_{\gamma,\varepsilon})\, a\, |V'|.$$

Therefore,

$$(1-\alpha_{\gamma,\varepsilon})\, a\, |V'| \ < \ |\mathcal{S}_B| \ \leq \ \frac{1}{2(1-\gamma)}|V'|\, a_B^{\max},$$

which implies

$$a_B^{\max} \ > \ 2(1-\gamma)(1-\alpha_{\gamma,\varepsilon})a. \tag{A.8}$$

The set $\mathcal{S}_W$ satisfies

$$|\mathcal{S}_W| \ = \sum_{v \in V' \cap W} \sum_{i \in I} \mathbb{1}_{X^i}(v) \ = \sum_{v \in V' \cap W} |I_v| \ \overset{\text{(A.7)}}{\geq} \ |V' \cap W|\, a_W^{\min}$$
$$\geq \ (|W| - |V \setminus V'|)\, a_W^{\min} \ \overset{\text{(A.1)}}{\geq} \ \left(\tfrac{1}{2} - \gamma\right)|V|\, a_W^{\min} \ \geq \ \left(\tfrac{1}{2} - \gamma\right)|V'|\, a_W^{\min}$$

and

$$|\mathcal{S}_W| \;=\; \sum_{i\in I}\sum_{v\in V'\cap W} \mathbb{1}_{X^i}(v) \;=\; \sum_{i\in I}|X^i\cap W| \;\overset{(A.5)}{<}\; \sum_{i\in I}\alpha_{\gamma,\varepsilon}|X^i| \;\overset{(A.2)}{=}\; \alpha_{\gamma,\varepsilon}a|V'|.$$

Consequently,

$$\left(\tfrac{1}{2}-\gamma\right)|V'|\,a_W^{\min} \;\le\; |\mathcal{S}_W| \;<\; \alpha_{\gamma,\varepsilon}a\,|V'|\,,$$

which implies that

$$a_W^{\min} \;<\; \frac{\alpha_{\gamma,\varepsilon}}{\left(\frac{1}{2}-\gamma\right)}a. \tag{A.9}$$

The remainder is now reduced to pure computation. Recall the definition of $\alpha_{\gamma,\varepsilon}$ in (A.4), that is

$$\alpha_{\gamma,\varepsilon} := \frac{(1-2\gamma)(1-\gamma-\varepsilon)}{(1-\gamma)(1-2\gamma)+1}.$$

This is equivalent to

$$\begin{aligned}
& (1-\gamma)(1-2\gamma)-\varepsilon(1-2\gamma) &=&\; \alpha_{\gamma,\varepsilon}(1-\gamma)(1-2\gamma)+\alpha_{\gamma,\varepsilon}\\
\Leftrightarrow\quad & (1-\gamma)(1-2\gamma)(1-\alpha_{\gamma,\varepsilon})-\alpha_{\gamma,\varepsilon} &=&\; \varepsilon(1-2\gamma)\\
\Leftrightarrow\quad & 2(1-\gamma)(1-\alpha_{\gamma,\varepsilon})-\frac{\alpha_{\gamma,\varepsilon}}{\left(\frac{1}{2}-\gamma\right)} &=&\; 2\varepsilon. \qquad\qquad (A.10)
\end{aligned}$$

Finally,

$$\begin{aligned}
2\varepsilon a \;&=\; (1+\varepsilon)a-(1-\varepsilon)a \;\overset{(A.3)}{\ge}\; \max\{|I_v| : v\in V'\}-\min\{|I_v| : v\in V'\}\\
&\ge\; a_B^{\max}-a_W^{\min} \;\overset{(A.8),(A.9)}{>}\; 2(1-\gamma)(1-\alpha_{\gamma,\varepsilon})a-\frac{\alpha_{\gamma,\varepsilon}}{\left(\frac{1}{2}-\gamma\right)}a \;\overset{(A.10)}{=}\; 2\varepsilon a,
\end{aligned}$$

which is a contradiction. Hence, there is an index $i_W\in I$ that satisfies $|X^{i_W}\cap W|\ge\alpha_{\gamma,\varepsilon}|X^i|$. □

Consider a (γ,ε,k,c)-path-prosperous graph G. Let $\mathcal{X}$ be a collection of clusters that shows that G satisfies Definition A.1. The previous lemma states that, for every bisection (B,W) in G, there exists a cluster with many vertices in B and there exists a cluster with many vertices in W. Then, (P3) guarantees many edge-disjoint paths that start in B and end in W, which will now be used to derive a lower bound for the width of a minimum bisection in G.

Theorem A.3.
For every $0\le\gamma<\frac{1}{2}$ and for every $0\le\varepsilon<1-\gamma$ there is an $\alpha_{\gamma,\varepsilon}>0$ such that the following holds. For every (γ,ε,k,c)-path-prosperous graph $G=(V,E)$ with $|V|$ even

$$\mathrm{MinBis}(G)\ge\alpha_{\gamma,\varepsilon}\cdot ck,$$

where $\alpha_{\gamma,\varepsilon}$ can be chosen as in Lemma A.2.

Proof. Fix some $0\le\gamma<\frac{1}{2}$ and $0\le\varepsilon<1-\gamma$. Let $\alpha_{\gamma,\varepsilon}$ be as in Lemma A.2. Moreover, let $G=(V,E)$ be an arbitrary (γ,ε,k,c)-path-prosperous graph with $|V|$ even. Then, there is a subgraph G' and a collection $\mathcal{X}=(X^i)_{i\in I}$ of clusters with $X^i\subseteq V(G')$ for all $i\in I$ with the properties in Definition A.1. Consider a minimum bisection (B,W) in G. By Lemma A.2 there exist indices i_B and i_W with $|X^{i_B}\cap B|\ge$

$\alpha_{\gamma,\varepsilon}|X^{i_B}|$ and $|X^{i_W} \cap W| \geq \alpha_{\gamma,\varepsilon}|X^{i_W}|$. Using (P2), it follows that $|X^{i_B} \cap B| \geq \alpha_{\gamma,\varepsilon}k$ and $|X^{i_W} \cap W| \geq \alpha_{\gamma,\varepsilon}k$. Moreover, define

$$d \;:=\; \min\left\{\left|X^{i_B} \cap B\right|, \left|X^{i_W} \cap W\right|, k\right\},$$

which satisfies $\alpha_{\gamma,\varepsilon}k \leq d \leq k$. Now, choose $Z_B \subseteq X^{i_B} \cap B$ and $Z_W \subseteq X^{i_W} \cap W$ with $|Z_B| = |Z_W| = d$. Then, (P3) implies that there are at least cd edge-disjoint Z_B,Z_W-paths in G' and, hence, also in G. Each such path has length at least one, i. e., it contains at least one edge, since $Z_B \cap Z_W \subseteq B \cap W = \emptyset$. Consequently, each path contains at least one edge that is cut by the bisection (B, W). Thus $e_G(B, W) \geq cd \geq \alpha_{\gamma,\varepsilon}ck$.$\square$

Appendix B

Embeddings of the Grid and Minimal Graphs Containing a Grid as a Minor

Here, a proof for Remark 3.15, which is restated in the following, is presented.

Remark B.1 (Remark 3.15 repeated).
For every integer $k \geq 3$, every minimal graph containing a $k \times k$ grid as a minor is uniquely embeddable.

Instead of working with embeddings of graphs, the following theorem is employed. Recall that a graph G is uniquely embeddable if, for any two embeddings of G in the plane, there is a topological isomorphism between the embeddings.

Theorem B.2 (Whitney, see Theorem 4.3.2 in [Die12]).
Every 3-connected, planar graph is uniquely embeddable.

Consequently, it would suffice to show that, for $k \geq 3$, every minimal graph G containing a $k \times k$ grid as a minor is 3-connected. However, even the $k \times k$ grid itself is not 3-connected, as can be seen by removing the vertices $(1, 2)$ and $(2, 1)$, which results in the vertex $(1, 1)$ being isolated. In general, every graph on at least four vertices with a vertex of degree two cannot be 3-connected. However, when studying embeddings, vertices of degree two can be neglected in the following way. Consider a graph G that is uniquely embeddable, then every graph that is a subdivision of G is also uniquely embeddable.

Corollary B.3.
Every subdivision of a 3-connected planar graph is uniquely embeddable.

The reverse operation of subdividing an edge is to *suppress a vertex of degree* 2. More formally, let $G = (V, E)$ be a graph and let $v \in V$ be a vertex with $\deg_G(v) = 2$ and such that the two neighbors of v, say u and w, are not adjacent. Then, the graph obtained from G by suppressing v is the graph, that is obtained from G by deleting the vertex v and inserting the edge $\{u, w\}$.

As usual, denote the $k \times k$ grid by G_k. For $k \geq 3$, the vertices $(1,1)$, $(1,k)$, $(k,1)$, and (k,k) are referred to as the *corners* of G_k. For $k \geq 3$, let G'_k be the graph obtained from G_k by suppressing each of its corners. For $i \in [k]$, the *i^{th} column* of G'_k is the vertex set obtained by intersecting the i^{th} column of G_k with $V(G'_k)$ and similarly, for $j \in [k]$, the *j^{th} row* of G'_k is the vertex set obtained by intersecting the j^{th} row of G_k with $V(G'_k)$. Furthermore, an edge $e \in E(G'_k)$ is called a *horizontal* edge if e is a horizontal edge of G_k and $e \in E(G'_k)$ is called a *vertical* edge if e is a vertical edge of G_k. Observe that G'_k contains exactly four edges that are neither vertical nor horizontal.

One way to show that G'_k is 3-connected is to use the following variant of Tutte's Wheel Theorem. In this context, a contraction of an edge $e = \{v, w\}$ in a graph G is called a *Tutte contraction* if $\deg_G(v) \geq 3$ and $\deg_G(w) \geq 3$. A sequence of graphs $(\tilde{G}_1, \ldots, \tilde{G}_d)$ is called a *sequence of Tutte contractions from $\tilde{G}_1$ to $\tilde{G}_d$* if, for all $h \in [d-1]$, the graph $\tilde{G}_{h+1}$ is obtained from $\tilde{G}_h$ by a Tutte contraction. For simplicity, a Tutte contraction-sequence from a graph G to a graph isomorphic to K_4 is called a *complete Tutte contraction-sequence for G*.

Theorem B.4 (variant of Tutte's Wheel Theorem, see Theorem 3.2.5 in [Die12]).
A graph G is 3-connected if and only if there exists a complete Tutte contraction-sequence for G.

Now, the previous theorem will be used to prove that, for $k \geq 3$, the graph G'_k is 3-connected. Hence, for $k \geq 3$, the $k \times k$ grid is uniquely embeddable.

Lemma B.5.
The graph G'_k is 3-connected for all $k \geq 3$.

Proof. Recall that $V(G_k) = \{(i,j) \colon i \in [k], j \in [k]\}$ and therefore, for $k \geq 3$,

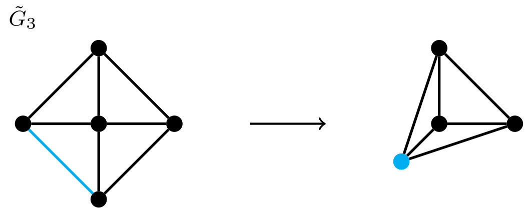
$$V(G'_k) = \{(i,j) \colon i \in [k], j \in [k]\} \setminus \{(1,1),(1,k),(k,1),(k,k)\}.$$

By Theorem B.4, it suffices to show that, for every integer $k \geq 3$, there is a complete Tutte contraction-sequence for G'_k. This is done by induction on k.

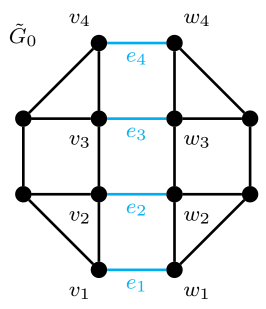
Base: The base is at $k = 3$. Set $G = G'_3$ and note that $\deg_G((1,2)) = \deg_G((2,1)) = 3$. Furthermore, $(1,2)$ and $(2,1)$ are adjacent in G and, when contracting the edge between them, a complete graph $\tilde{G}$ on four vertices is obtained, see Figure B.1a). Hence, $(G, \tilde{G})$ is a complete Tutte contraction-sequence for G.

Step: Fix some $k \geq 4$ and assume that there exists a complete Tutte contraction-sequence for G'_{k-1}. In order to show that there is a complete Tutte contraction-sequence for G'_k, it suffices to show that there is a Tutte contraction-sequence from G'_k to a graph isomorphic to G'_{k-1}. The idea is to contract all horizontal edges between the second and the third column of G'_k and then all vertical edges between the second and the third row of the remaining graph, see Figure B.1c) for an example. It is now argued that, when contracting the horizontal edges between the second and the third column of G'_k successively, then each contraction is a Tutte contraction. Arguing that each contraction of a vertical edge is a Tutte contraction is analogous.

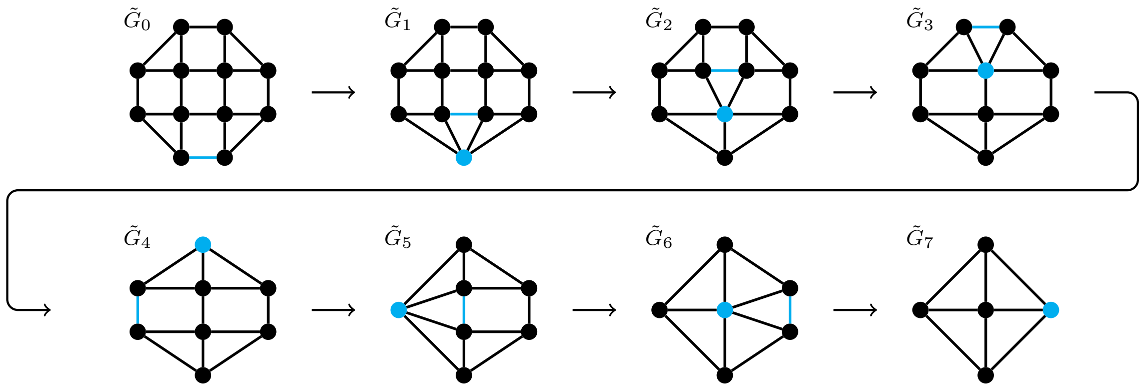
For $h \in [k]$, set $v_h := (2,h)$, $w_h := (3,h)$, and $e_h := \{v_h, w_h\}$, see Figure B.1b). Define $\tilde{G}_0 := G'_k$ and, for $h \in [k]$, let $\tilde{G}_h$ be the graph obtained from $\tilde{G}_{h-1}$ by contracting e_h. As $\deg_{\tilde{G}_0}(v_1) = \deg_{\tilde{G}_0}(w_1) = 3$, the contraction of e_1 in $\tilde{G}_0$ is a Tutte contraction. For $h \in [k-1] \setminus \{1\}$, in the graph $\tilde{G}_{h-1}$, the vertex $v_h = (2,h)$ is adjacent to all of $(1,h)$, $(2,h+1)$, $w_h = (3,h)$, and the vertex resulting from the contraction of e_{h-1}, which implies that $\deg_{\tilde{G}_{h-1}}(v_h) = 4$. Similarly, it follows that $\deg_{\tilde{G}_{h-1}}(w_h) = 4$ and



a) Contraction in the base case, i. e., for $k = 3$.



b) Notation in the step, example with $k = 4$.



c) Example for contractions as involved in the step, example with $k = 4$.

Figure B.1: Contractions in the proof of Lemma B.5. Right before contracting an edge it is colored blue and right after the contraction the resulting vertex is colored blue.

therefore the contraction of e_h in $\tilde{G}_{h-1}$ is a Tutte contraction. Moreover, in $\tilde{G}_{k-1}$, the vertex $v_k = (2, k)$ is adjacent to all of $(1, k-1)$, $w_k = (3, k)$, and the vertex resulting from the contraction of e_{k-1}, which implies that $\deg_{\tilde{G}_{k-1}}(v_k) = 3$. Similarly, it follows that $\deg_{\tilde{G}_{k-1}}(w_k) = 3$ and therefore the contraction of e_k in $\tilde{G}_{k-1}$ is a Tutte contraction. □

Using Lemma B.5, we can now prove Remark B.1.

Proof of Remark B.1. Let $k \geq 3$. Let $H = (V, E)$ be a minimal graph that contains a $k \times k$ grid as a minor. Denote the $k \times k$ grid by G_k, and let G'_k be the graph obtained from G_k by suppressing all of its corners. Consider a partition of V into non-empty sets $M_{i,j}$ with $i, j \in [k]$ such that contracting $M_{i,j}$ to one vertex (i, j) yields G_k, see also Proposition 3.24. As long as H contains a vertex v with $\deg_H(v) = 2$ such that the neighbors of v are not adjacent, suppress v. Denote by $H' = (V', E')$ the graph obtained from H in this way. Clearly, H is a subdivision of H'. Furthermore, all vertices $v \in V'$ satisfy $\deg_{H'}(v) = \deg_H(v)$. Assume for a contradiction that H' contains a vertex v with $\deg_{H'}(v) = 2$. Then, v was not suppressed because its two neighbors in H', say u and w, are adjacent. Thus, (u, v, w) must be a cycle in H'. This implies that H contains a cycle C such that at most two vertices x in C satisfy $\deg_H(x) \geq 3$. When contracting H to G_k, the cycle C cannot be contracted to a single vertex of G_k as otherwise there would be $i, j \in [k]$ with $V(C) \subseteq M_{i,j}$. Thus, contradicting Proposition 3.24a). Consequently, when contracting H to G_k, there must be a cycle in G_k that contains at most two vertices x with $\deg_{G_k}(x) \geq 3$, which is again a contradiction. Consequently, H' does not contain a vertex v with $\deg_{H'}(v) = 2$.

For $i, j \in [k]$, set $M'_{i,j} = M_{i,j} \cap V'$. Clearly, the sets $M'_{i,j}$ with $i, j \in [k]$ form a partition of V'. For all $i, j \in [k]$ such that (i, j) is a vertex of G'_k, the set $M'_{i,j}$ is non-empty by Proposition 3.16. If $v, w \in V$ are two vertices with $\deg_H(v) \geq 3$ and $\deg_H(w) \geq 3$, then v and w must both be vertices of H' and, for each v,w-path P in H, there exists a v,w-path P' in H' with

$$\left\{(i, j)\colon\ P' \text{ uses a vertex from } M'_{i,j}\right\} \subseteq \{(i, j)\colon\ P \text{ uses a vertex from } M_{i,j}\}\,.$$

Thus, Proposition 3.24 implies the following.

(i) For all $i, j \in [k]$ such that (i, j) is a vertex of G'_k, the graph $H'[M'_{i,j}]$ is connected.

(ii) For all $i, j, \tilde{i}, \tilde{j} \in [k]$ such that (i, j) and $(\tilde{i}, \tilde{j})$ are adjacent vertices of G'_k, there is exactly one edge in H' that joins a vertex in $M'_{i,j}$ to a vertex in $M'_{\tilde{i},\tilde{j}}$.
For all $i, j, \tilde{i}, \tilde{j} \in [k]$ such that (i, j) and $(\tilde{i}, \tilde{j})$ are vertices of G'_k that are not adjacent, there is no edge in H' that joins a vertex in $M'_{i,j}$ to a vertex in $M'_{\tilde{i},\tilde{j}}$.

Therefore, when taking H' and contracting each set $M'_{i,j}$ to one vertex (i, j) for all $i, j \in [k]$ such that (i, j) is a vertex of G'_k yields G'_k. Let $i, j \in [k]$ such that (i, j) is a vertex of G'_k. Proposition 3.16 now implies that either $|M'_{i,j}| = 1$ or $|M'_{i,j}| = 2$ and in the latter case each vertex $v \in M'_{i,j}$ satisfies $\deg_{H'}(v) = 3$. Hence, if $|M'_{i,j}| \geq 2$, then a contraction of $M'_{i,j}$ in H' to a single vertex is a Tutte contraction. Moreover, when contracting $M'_{i,j}$ in H', the degree of each vertex not in $M'_{i,j}$ does not decrease due to (ii).

All in all, there exists a sequence of Tutte contractions from H' to G'_k, which can be extended to a complete Tutte contraction-sequence for H' by Lemma B.5. Therefore, it follows from Theorem B.4 that H' is 3-connected. Recalling that H is a subdivision of H' now implies that H is uniquely embeddable by Corollary B.3. □

Appendix B

Bibliography

[ACP87] S. Arnborg, D. G. Corneil, and A. Proskurowski. “Complexity of Finding Embeddings in a k-Tree”. In: *SIAM Journal on Algebraic Discrete Methods* 8.2 (1987), pp. 277–284.

[AKK99] S. Arora, D. Karger, and M. Karpinski. “Polynomial Time Approximation Schemes for Dense Instances of NP-Hard Problems”. In: *Journal of Computer and System Sciences* 58.1 (1999), pp. 193–210.

[AR06] K. Andreev and H. Räcke. “Balanced Graph Partitioning”. In: *Theory of Computing Systems* 39.6 (2006), pp. 929–939.

[BBG11] S. Borgwardt, A. Brieden, and P. Gritzmann. “Constrained Minimum-k-Star Clustering and its Application to the Consolidation of Farmland”. In: *Operational Research* 11.1 (2011), pp. 1–17.

[BK02] P. Berman and M. Karpinski. “Approximation Hardness of Bounded Degree MIN-CSP and MIN-BISECTION”. In: *Automata, Languages and Programming.* Vol. 2380. Lecture Notes in Computer Science. Berlin: Springer, 2002, pp. 623–632.

[BK10] M. de Berg and A. Khosravi. “Optimal Binary Space Partitions in the Plane”. In: *Computing and Combinatorics: 16th Annual International Conference, COCOON 2010, Nha Trang, Vietnam, July 19-21, 2010. Proceedings.* Ed. by M. T. Thai and S. Sahni. Berlin, Heidelberg: Springer, 2010, pp. 216–225.

[BL84] S. N. Bhatt and F. T. Leighton. “A Framework for Solving VLSI Graph Layout Problems”. In: *Journal of Computer and System Sciences* 28.2 (1984), pp. 300 –343.

[BL91] B. Bollobás and I. Leader. “Edge-Isoperimetric Inequalities in the Grid”. In: *Combinatorica* 11.4 (1991), pp. 299–314.

[Bod96] H. L. Bodlaender. “A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth”. In: *SIAM Journal on Computing* 25.6 (1996), pp. 1305–1317.

[Bod98] H. L. Bodlaender. “A Partial k-Arboretum of Graphs with Bounded Treewidth”. In: *Theoretical Computer Science* 209.1-2 (1998), pp. 1–45.

[Bui+87] T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser. “Graph Bisection Algorithms with Good Average Case Behavior”. In: *Combinatorica* 7.2 (1987), pp. 171–191.

[Bul+02] R. Bulterman, F. van der Sommen, G. Zwaan, T. Verhoeff, A. van Gasteren, and W. Feijen. “On Computing a Longest Path in a Tree”. In: *Information Processing Letters* 81.2 (2002), pp. 93–96.

[Cor+09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. 3rd ed. MIT Press, 2009.

[CW90] E. R. Canfield and S. G. Williamson. “The Two Basic Linear Time Planarity Algorithms: Are They the Same?” In: *Linear and Multilinear Algebra* 26.4 (1990), pp. 243–265.

[Cyg+14] M. Cygan, D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. “Minimum Bisection is Fixed Parameter Tractable”. In: *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*. STOC ’14. New York, NY, USA: ACM, 2014, pp. 323–332.

[DB13] M. De Biasi. *Complexity of Finding Large Grid Minors*. 2013. URL: http://cstheory.stackexchange.com/questions/18301/complexity-of-finding-large-grid-minors (visited on 11/02/2016).

[Die12] R. Diestel. *Graph Theory*. 4th ed. Vol. 173. Graduate texts in mathematics. Springer-Verlag, Heidelberg, 2012.

[Dik+93] K. Diks, H. N. Djidjev, O. Sykora, and I. Vrto. “Edge Separators of Planar and Outerplanar Graphs With Applications”. In: *Journal of Algorithms* 14.2 (1993), pp. 258 –279.

[Dji82] H. N. Djidjev. “On the Problem of Partitioning Planar Graphs”. In: *SIAM Journal on Algebraic Discrete Methods* 3.2 (1982), pp. 229–240.

[DM14] J. Díaz and G. B. Mertzios. “Minimum Bisection is NP-hard on Unit Disk Graphs”. In: *CoRR* abs/1404.0117 (2014).

[DV97] H. N. Djidjev and S. M. Venkatesan. “Reduced Constants for Simple Cycle Graph Separation”. In: *Acta Inform.* 34.3 (1997), pp. 231–243.

[ELM03] R. Elsässer, T. Lücking, and B. Monien. “On Spectral Bounds for the k-Partitioning of Graphs”. In: *Theory of Computing Systems* 36.5 (2003), pp. 461–478.

[Fel13] A. E. Feldmann. “Fast Balanced Partitioning is Hard Even on Grids and Trees”. In: *Theoretical Computer Science* 485 (2013), pp. 61 –68.

[FF15] A. E. Feldmann and L. Foschini. “Balanced Partitions of Trees and Applications”. In: *Algorithmica* 71.2 (2015), pp. 354–376.

[FHL08] U. Feige, M. Hajiaghayi, and J. R. Lee. “Improved Approximation Algorithms for Minimum Weight Vertex Separators”. In: *SIAM Journal on Computing* 38.2 (2008), pp. 629–657.

[FK02] U. Feige and R. Krauthgamer. “A Polylogarithmic Approximation of the Minimum Bisection”. In: *SIAM Journal on Computing* 31.4 (2002), pp. 1090–1118.

[FST13] C. G. Fernandes, T. J. Schmidt, and A. Taraz. “On the Structure of Graphs with Large Minimum Bisection”. In: *The Seventh European Conference on Combinatorics, Graph Theory and Applications*. Ed. by J. Nešetřil and M. Pellegrini. Vol. 16. CRM Series. Scuola Normale Superiore, 2013, pp. 291–296.

[FST15a] C. G. Fernandes, T. J. Schmidt, and A. Taraz. “On Minimum Bisection and Related Partition Problems in Graphs with Bounded Tree Width”. In: *Electronic Notes in Discrete Mathematics* 49 (2015), pp. 481–488.

[FST15b] C. G. Fernandes, T. J. Schmidt, and A. Taraz. “Approximating Minimum k-Section in Trees with Linear Diameter”. In: *Electronic Notes in Discrete Mathematics* 50 (2015), pp. 71–76.

[FSTa] C. G. Fernandes, T. J. Schmidt, and A. Taraz. “On Minimum Bisection and Related Cut Problems in Trees and Tree-Like Graphs”. Submitted to the Journal of Graph Theory.

[FSTb] C. G. Fernandes, T. J. Schmidt, and A. Taraz. “Approximating Minimum k-Section in Trees with Linear Diameter and an Extension to Tree-Like Graphs”. In preparation.

[FSTc] C. G. Fernandes, T. J. Schmidt, and A. Taraz. “Structural Results and Algorithms for Planar Graphs with Large Minimum Bisection Width”. In preparation.

[FW15] A. E. Feldmann and P. Widmayer. “An $O(n^4)$ Time Algorithm to Compute the Bisection Width of Solid Grid Graphs”. In: *Algorithmica* 71.1 (2015), pp. 181–200.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[GJS76] M. R. Garey, D. S. Johnson, and L. Stockmeyer. “Some Simplified NP-Complete Graph Problems”. In: *Theoretical Computer Science* 1.3 (1976), pp. 237–267.

[Gol78] E. M. Gold. “Deadlock Prediction: Easy and Difficult Cases”. In: *SIAM Journal on Computing* 7.3 (1978), pp. 320–336.

[Gri11] A. Grigoriev. “Tree-Width and Large Grid Minors in Planar Graphs”. In: *Discrete Mathematics & Theoretical Computer Science* 13.1 (2011).

[GT08] Q.-P. Gu and H. Tamaki. “Optimal Branch-Decomposition of Planar Graphs in $\mathcal{O}(n^3)$ Time”. In: *ACM Trans. Algorithms* 4.3 (2008), 30:1–30:13.

[GT11] Q.-P. Gu and H. Tamaki. “Constant-Factor Approximations of Branch-Decomposition and Largest Grid Minor of Planar Graphs in $\mathcal{O}(n^{1+\varepsilon})$ Time”. In: *Theoretical Computer Science* 412.32 (2011), pp. 4100 –4109.

[Ham16] F. Hamann. “Über kleinste Bisektionen und k-Sektionen in gewichteten Bäumen”. German. Bachelor’s Thesis. Germany: Technische Universität Hamburg, 2016.

[HK73] J. E. Hopcroft and R. M. Karp. “An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs”. In: *SIAM Journal on Computing* 2.4 (1973), pp. 225–231.

[HO92] J. Hao and J. B. Orlin. “A Faster Algorithm for Finding the Minimum Cut in a Graph”. In: *SODA ’92: Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992, pp. 165–174.

[HT74] J. Hopcroft and R. E. Tarjan. “Efficient Planarity Testing”. In: *Journal of the Association for Computing Machinery* 21.4 (1974), pp. 549–568.

[Jan+05] K. Jansen, M. Karpinski, A. Lingas, and E. Seidel. “Polynomial Time Approximation Schemes for MAX-BISECTION on Planar and Geometric Graphs”. In: *SIAM Journal on Computing* 35.1 (2005), pp. 110–119.

[KNS09] R. Krauthgamer, J. S. Naor, and R. Schwartz. “Partitioning Graphs into Balanced Components”. In: *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’09. New York: Society for Industrial and Applied Mathematics, 2009, pp. 942–949.

[KR92] D. E. Knuth and A. Raghunathan. “The Problem of Compatible Representatives”. In: *SIAM J. Discret. Math.* 5.3 (1992), pp. 422–427.

[KT06] J. Kleinberg and É. Tardos. *Algorithm Design.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[LEC67] A. Lempel, S. Even, and I. Cederbaum. “An Algorithm for Planarity Testing of Graphs”. In: *Theory of Graphs.* Ed. by P. Rosenstiehl. New York: Gordon and Breach, 1967, pp. 215–232.

[Lei92] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Array, Trees, Hypercubes.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992.

[Lic82] D. Lichtenstein. “Planar Formulae and their Uses”. In: *SIAM Journal on Computing* 11.2 (1982), pp. 329–343.

[LST90] J. K. Lenstra, D. B. Shmoys, and É. Tardos. “Approximation Algorithms for Scheduling Unrelated Parallel Machines”. In: *Mathematical Programming* 46.1 (1990), pp. 259–271.

[LT79] R. J. Lipton and R. E. Tarjan. “A Separator Theorem for Planar Graphs”. In: *SIAM Journal on Applied Mathematics* 36.2 (1979), pp. 177–189.

[Moh92] B. Mohar. “Laplace Eigenvalues of Graphs - a Survey”. In: *Discrete Mathematics* 109.1-3 (1992), pp. 171–183.

[MT01] B. Mohar and C. Thomassen. *Graphs on Surfaces.* Johns Hopkins studies in the mathematical sciences. Baltimore: The Johns Hopkins University Press, 2001.

[PS96] C. H. Papadimitriou and M. Sideri. “The Bisection Width of Grid Graphs”. In: *Mathematical Systems Theory* 29.2 (1996), pp. 97–110.

[Ree97] B. A. Reed. “Tree Width and Tangles: A New Connectivity Measure and Some Applications”. In: *Surveys in Combinatorics, 1997.* Ed. by R. Bailey. Cambridge University Press, 1997, pp. 87–162.

[RLWW97] H. Ripphausen-Lipa, D. Wagner, and K. Weihe. “The Vertex-Disjoint Menger Problem in Planar Graphs”. In: *SIAM Journal on Computing* 26.2 (1997), pp. 331–349.

[RST94] N. Robertson, P. D. Seymour, and R. Thomas. “Quickly Excluding a Planar Graph”. In: *Journal of Combinatorial Theory, Series B* 62.2 (1994), pp. 323–348.

[Räc08] H. Räcke. “Optimal Hierarchical Decompositions for Congestion Minimization in Networks”. In: *Proceedings of the 40th Annual ACM Symposium on Theory of Computing.* STOC 2008. Victoria, British Columbia, Canada: ACM, 2008, pp. 255–264.

[Sch13] T. J. Schmidt. *Bisecting Trees and Planar Graphs.* Master’s thesis. Technische Universität München, Germany, 2013.

[Shm97] D. B. Shmoys. “Approximation Algorithms for Cut Problems and their Application to Divide-and-Conquer”. In: *Approximation Algorithms for NP-hard Problems.* Ed. by D. S. Hochbaum. Boston, MA, USA: PWS Publishing Co., 1997, pp. 192 –235.

[ST97] H. D. Simon and S.-H. Teng. “How Good is Recursive Bisection?” In: *SIAM Journal on Scientific Computing* 18.5 (1997), pp. 1436–1445.

[SW11] R. Sedgewick and K. Wayne. *Algorithms.* 4th ed. Addison Wesley, 2011.

[Ye01] Y. Ye. “A .699-Approximation Algorithm for Max-Bisection”. In: *Mathematical Programming* 90.1 (2001), pp. 101–111.